DERIVATION OF LINE CHARGE ALGORITHMS

FOR

ALPINE ENERGY LIMITED

A.L.P. – ALPINE LINE-CHARGE PROGRAM.

A Report Submitted in partial fulfillment of the Master of Engineering Management degree at the University of Canterbury, Christchurch, New Zealand.

By

CHRISTOPHER SAVAGE

UNIVERSITY OF CANTERBURY

DECEMBER, 1997

1) Acknowledgments

I would like to take some time at this point to express my thanks to all the people that have been involved in the successful completion of this project.

The project would not of been possible without the support and help from Jim Pearce, Robert Mann and the Drawing Office at Alpine Energy in Timaru. Jim Pearce for making the project possible in the first place and providing the momentum. Robert Mann for continually looking for the bugs in the software (and report) and proposing the features that make them both that much better.

My academic supervisor Dr Chris Arnold for the help and guidance I needed, and for his invaluable suggestions for this document. Grant Smith of ACE Computer Consultants Limited in Invercargill for understanding the bits that I didn't.

Last but not least, my friends, parents, flatmates and classmates for their help throughout the year.

Any mistakes that remain are my own.

2) Contents Page

3) Table of Figures

4) Executive Summary

This report documents the work performed during 1997 by the author developing and examining possible line charge algorithms to be applied to the Alpine Energy Limited electrical distribution network.

A database is created that enables three existing databases used by Alpine to be connected, allowing correlation of information across the databases. The most important effect of this is the ability to calculate the value of each segment in the network.

A software package is created that performs the above action and allows the capital cost recovery from the network (in the form of line charges) to be calculated using a series of over ten thousand possible asset-based and installation-based methods.

The software is highly adaptable and allows line charge investigations of a very diverse nature. Performance levels of assets may be calculated, along with identification of which customers in the network are being subsidised, and to what extent.

All files used by and created by the software are in an easy to read, text based format. In the case of program output files, Microsoft Excel is the recommended package for analysis.

In preparation for a possible move to a GIS based system at Alpine, the software has been designed to allow changes to the input files that would result from a move away from Gentrack without a major re-work to the software package. While the database created allows the asset register to be linked into a GIS environment.

Algorithms have been built into the software that automatically check the validity of the database created for this project, enabling relatively straight-forward updating of this database.

5) INTRODUCTION

This report documents the work performed during 1997 by the author developing and examining possible line charge algorithms to be applied to the Alpine Energy Limited electrical distribution network.

The goal was to find a way to allocate line charges based on asset usage and by doing so identify under-performing assets within the Alpine Energy network.

A background to the company, industry and project is followed by a discussion of the current method for allocating line charges and the new method that this project proposes.

The resources available section contains details of the databases available for the completion of this project and the first of two major tasks of the project – the connection of these databases.

An overview of the software developed for this project is presented. This section looks at the motivation for the software, methods used during development and the theory developed for the software implementation.

The performance of the software itself is examined, its strengths and known weaknesses. A brief discussion of the general trends observed the results from the software and implementation issues is presented. Finally, recommendations are made for future use and development of the software.

Appendices of this report include the software user manual, data conventions and procedures, software prototypes and an index.

A glossary of the terms used in this report is included as appendix E.

6) Background

6.1 The Deregulated Energy Industry

Until the late 1980s, the energy industry in New Zealand was government owned and operated. Over 80 local bodies controlled the distribution networks and residential users of electricity were heavily subsidised by commercial users.

This changed in 1995/96, first with the creation of ECNZ as a state owned enterprise, then with the further splitting of ECNZ to create Contact Energy in order to create competition in the energy generation market. Transpower was split from ECNZ in 1988.

Legislation provided for competition in the domestic energy market from the 1st of April 1993 and in the commercial market from the 1st of April 1994. However, at present there is only limited competition for very large commercial customers and apart from some pilot schemes there is still a technological barrier to competition in the domestic market place.

Alpine Energy was established on the 1st of July 1993. It was created by the merger of the South Canterbury Power Board and Timaru Electricity following the deregulation of the industry by the Energy Companies Act 1992 [3].

A major modification required for the change in market structure was the separation of line and energy charges (SOLEC). Until this time it was accepted that domestic users of electricity were subsidised by commercial and industrial users.

This separation of charges also lead to a separation of business. Alpine Energy does not sell power to consumers. This action is performed United Electricity, 25% owned by Alpine Energy. The reasoning behind this choice was that a single large energy buying group has more bargaining power in the deregulated market place than a single small entity.

United Electricity, and its North Island sister company Pacific Energy sell energy for a large number of line companies, as shown below in Figure 1.

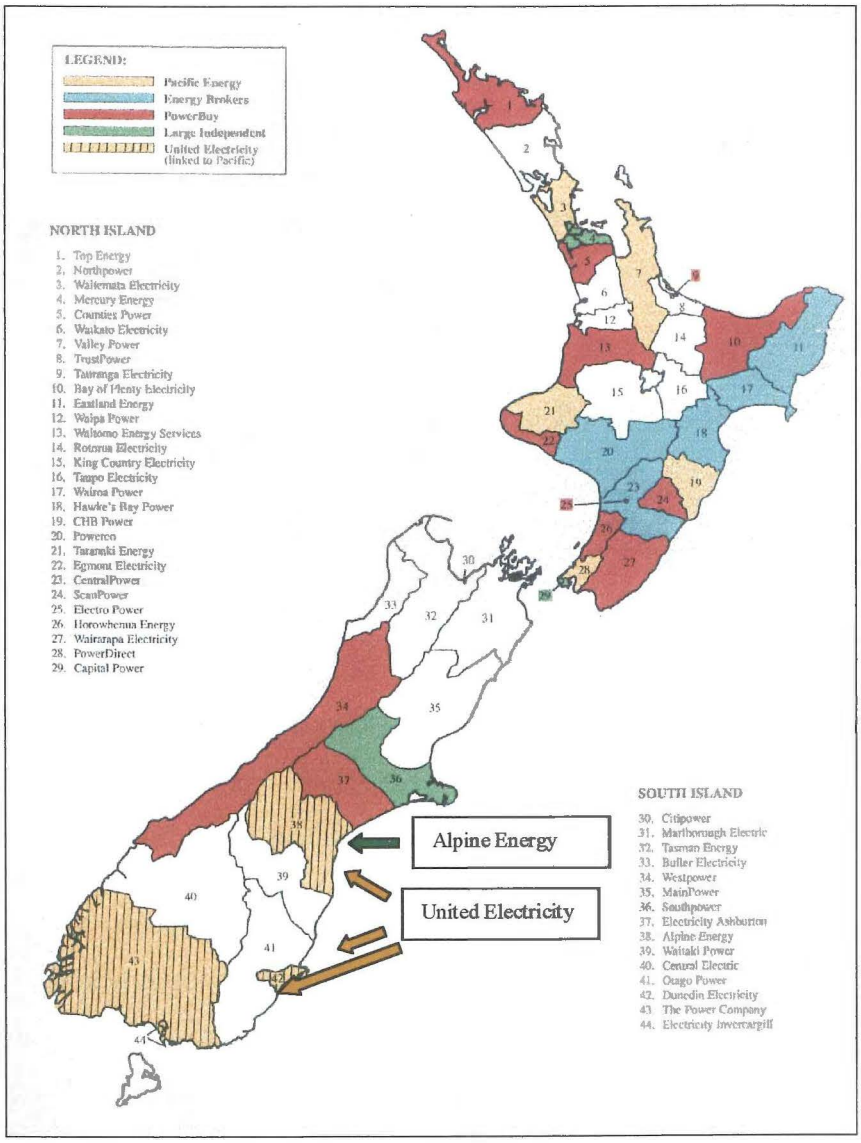

FIGURE 1 ENERGY BUYING GROUPS (1995)

The principal purpose of Alpine Energy Limited is to manage the operation and maintenance of its South Canterbury electricity distribution network. The company does not sell power to consumers, it provides a distribution network to energy suppliers. Accordingly, the income for the company is based on line charges.

A company distributing energy is now required by legislation to disclose information relating to "Pricing policies and methodologies [...] used for the allocation of [...] costs and revenues ..." [1]. A standardised method is set out in the legislation, but companies may choose to adopt an alternative pricing policy.

6.2 ALPINE ENERGY STATISTICS

For more detailed information on changes in the New Zealand energy industry, it is recommended that a copy of "The New Zealand Electricity Directory 1996" [2] is consulted. This document contains an excellent summary of the motivation for the energy industry deregulation and the changes during the 1980s and 1990s.

Also contained in this document [2] are the (1996) statistics on each of the 44 power companies. A sample from these statistics is provided below. The numbers quoted have been rounded for ease of reading.

- There are 1.6 million consumers in New Zealand, Alpine Energy supplies 26,000 (1.6%) making it the 14th largest supplier. The largest is Mercury Energy with 247,000 consumers and the smallest Buller Electricity with 4,000.

- Alpine supplies 441,500,000 kWhr of electricity annually. This is the 14th largest, but only 1.7% of the New Zealand total. Comparison with the number of consumers shows that South Canterbury network has an average demand placed on it from consumers .

- A total system length of 3,600 km is largely overhead line, although 400 km is underground. A total transformer capacity of 250,000 kVA is again the 14th largest in New Zealand, but only 10% that of Mercury Energy. This shows the very distributed nature of Alpine's consumers, especially taking into consideration that the Mercury network is only twice Alpine's length.

- Direct line costs are $781 per km for Alpine, the 7th lowest; alternatively $52 per customer, the 3rd lowest. The lowest are Central Hawkes Bay ($331 / km, but $150 / customer) and Dunedin electricity ($3,000 / km, but only $19 / customer).

6.3 PROJECT OBJECTIVES

The goal of the project is to examine customer line charges across the Alpine Energy distribution network, investigating differing allocations of the network operation costs to individual consumers.

These capital related costs include:

- The Transpower transmission cost
- A contribution towards the dividend for Alpine Energy shareholders
- The operating, maintenance and replacement cost of:
  - The substations
  - The high voltage network (33kV and 11kV)
  - Distribution transformers
  - The low voltage network (415V)

The project endeavours to utilise a fundamentally sound methodology to assign these costs to customers as line charges. Any methodology generated must satisfy Commerce Commission guidelines. Criteria are created and investigated, allowing a range of possible solutions to be developed and to allow evaluation of the "fairness" of each solution.

Criteria for the "fairness" of a solution included the impact on large and domestic customers, neighbours utilising different assets, and similar installations in different geographical areas.

In performing this project the transparency of the methods used were of paramount importance. A solution that could not be adequately defined was not acceptable.

A more detailed breakdown of objectives created in conjunction with Alpine Energy provides that the project and computer program developed must fulfil the following:

1. Investigate the relationship between customer network connections and the line charges allocated to the customer under current pricing strategies. These current strategies allocate a heavily averaged line charge to consumers.

2. A tool for identifying the relationship between asset utilisation and customer type. It was predicted that once a sufficient distance from the point of supply was reached, the current line charges would no longer recover the investment in assets required to transmit electrical energy to consumers.

3. A tool for investigating alternative line charging strategies based on asset utilisation or a combination of asset utilisation, delivered energy volume and fixed charges. This flexibility in driving factors is important for a full range of line charge strategies to be investigated.

4. To assess the performance of all major network assets and identify under-performing ones. This would be achieved by comparing the current revenue gathered from customers with the revenue that would be recovered under an asset-based scheme.

5. To provide a link between the existing customer database and the asset database. Thus alleviating the current inability to examine the cost recovery of the network from an asset-usage view point.

6. Program operations must be efficient to ensure that the information on 27,000 customers and 3,500 km of network assets would be readily transferable between computers and that analysis functions would run in a reasonable period of time. A program that takes more than 10-15 minutes to compute a scenario would be less attractive than one that can generate a result in a couple of minutes.

7. The program must accept changes in customer data, line charges and the amount to be recovered. A heavy emphasis on software flexibility is important for the future usefulness of the software.

8. The program must be capable of easy modification if assets or network points of supply are changed.

9. The program has to be capable of exporting results in a format capable of further manipulation in Excel spreadsheets and other Microsoft products.

10. The data used and created by the software must be understandable and verifiable if a manual audit is carried out.

11. The software must be Y2K compliant. A software package that is crippled by the year 2000 bug in only two years is not an attractive investment.

6.4 PRELIMINARY TASKS

Prior to the commencement of this project the author spent the summer working at Alpine Energy, becoming familiar with the methods of the company, their record-keeping systems and updating the Gentrack ENS database. This database forms a complete virtual representation of the physical layout and properties of the Alpine Energy HV distribution network.

The goals and scope of the project were developed and tuned over this period and a project proposal submitted in early March.

One of the first major tasks was a project kick-off meeting at Alpine Energy in April involving the Chief Executive and heads of all departments affected by this project. During this meeting the goal of the project from the MEM and Alpine point-of-view was discussed. This allowed all parties to understand the motivations behind the project.

A major decision made at the meeting was to create from scratch a totally customised database management package to achieve the goals of the project. This stemmed from the complex nature of computations involved and output required.

The basic structure of the finished product was developed and a request was made for the author to develop a timetable for the project. This timetable and work breakdown structure (WBS) proved invaluable in organising the project and allowing it to be completed on time.

This WBS is provided as Figure 2. Another use for the WBS was as a reporting tool. As each task were started, its border was changed to blue; completed tasks were coloured green and shaded.

The WBS allowed the project to be split into two distinctive tasks, data gathering and software development. Each of these will be examined in some detail after a brief overview of line charges in the following section.

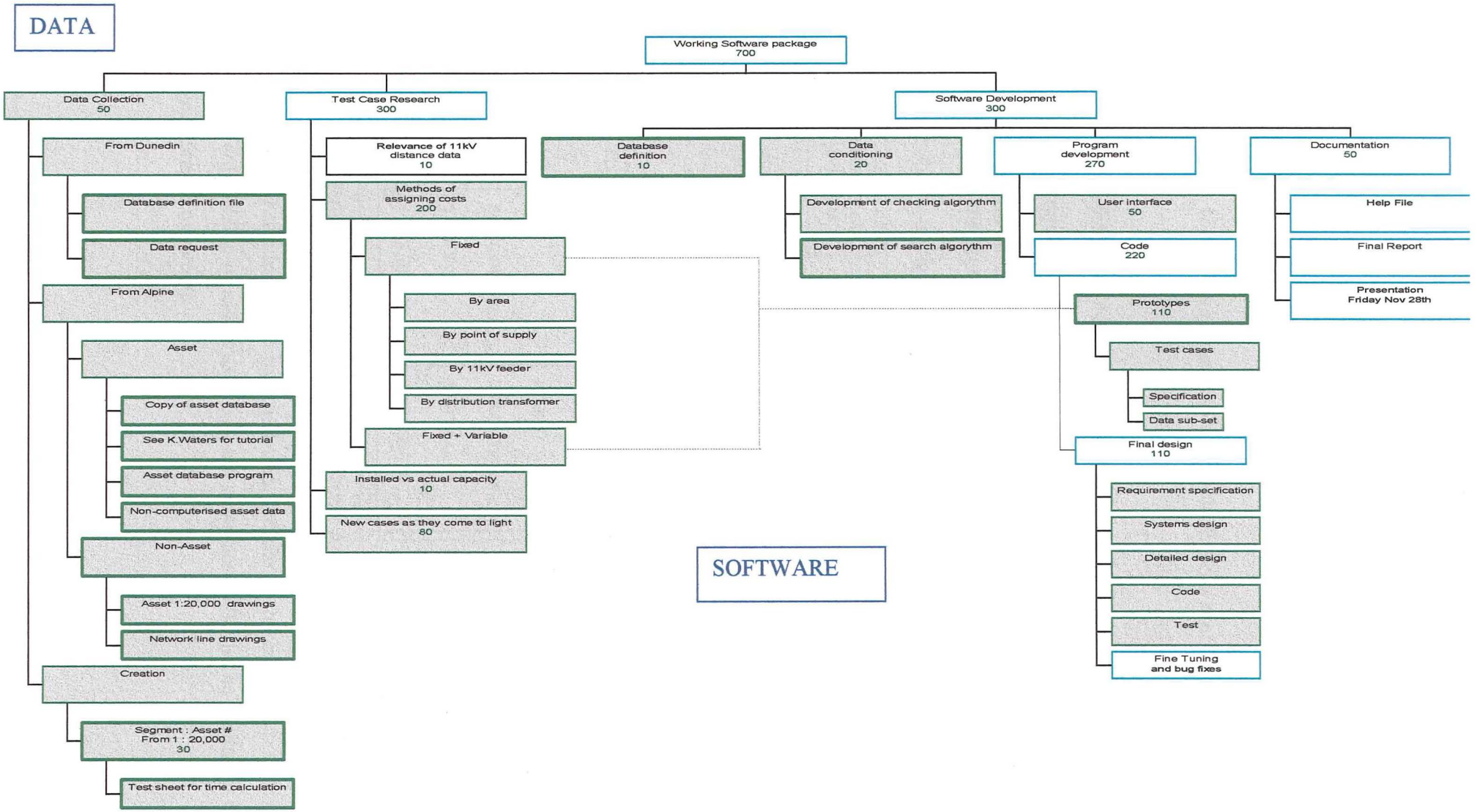

FIGURE 2 PROJECT WORK BREAKDOWN STRUCTURE

7) Line Charges

7.1 The Current Method

Currently the line charges implemented by Alpine Energy are calculated using the following method:

1. The annual costs that need to be recovered from the network are identified.
2. These are summed to give a total revenue requirement from the network.
3. This revenue is divided amongst five different consumer groups. Each member of the group pays the same line charge.

The example shown in Figure 3 uses actual Alpine Energy data from some time ago.

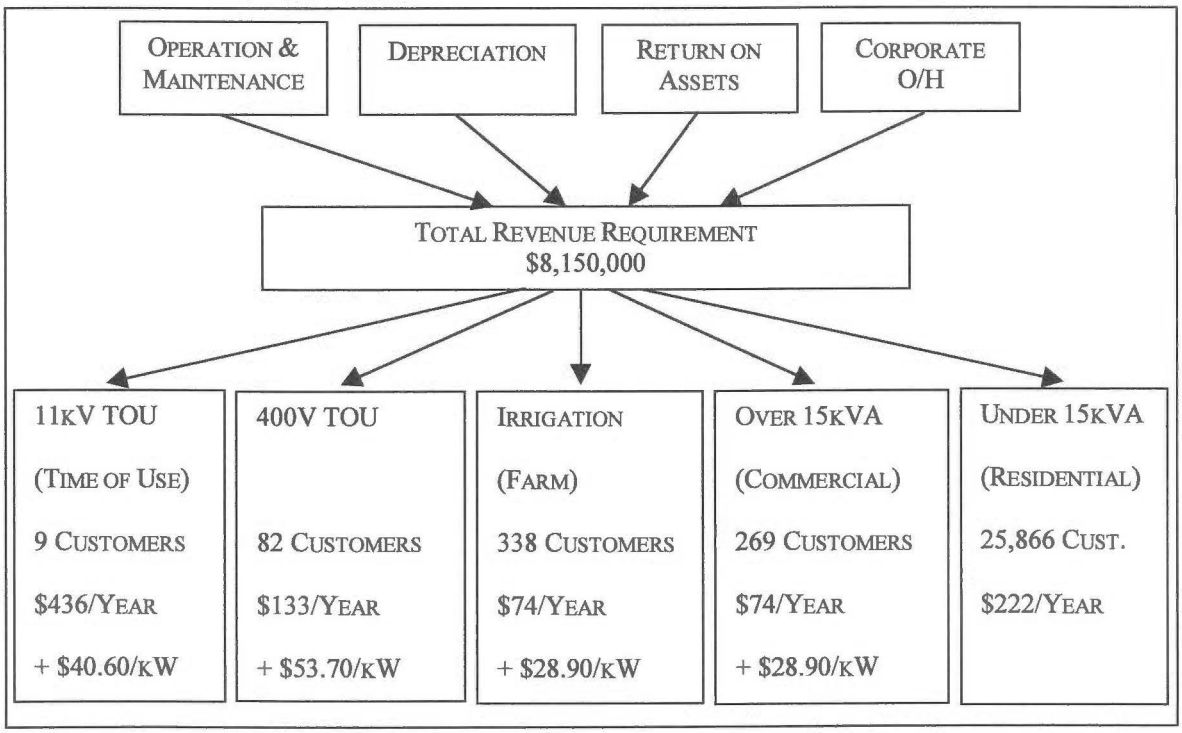

Figure 3 Allocation of Network Annual Costs.

This is the method recommended by legislation, but the averaging fails to take into account the dollar value of assets used by each customer. This is the limitation that this project aims to overcome.

7.2 The New Method

It was decided that following procedure would be developed for this project:

1. The same revenue requirement is used.
2. For each customer, the following statistics are generated:
   i) The fixed (kVA) and variable (kWhr) consumption,
   ii) the assets used (and type of asset valuation – ODV or REP),
   iii) and the type of customer (residential or large)
3. i, ii, and iii are each considered in order to generate an individual line charge.

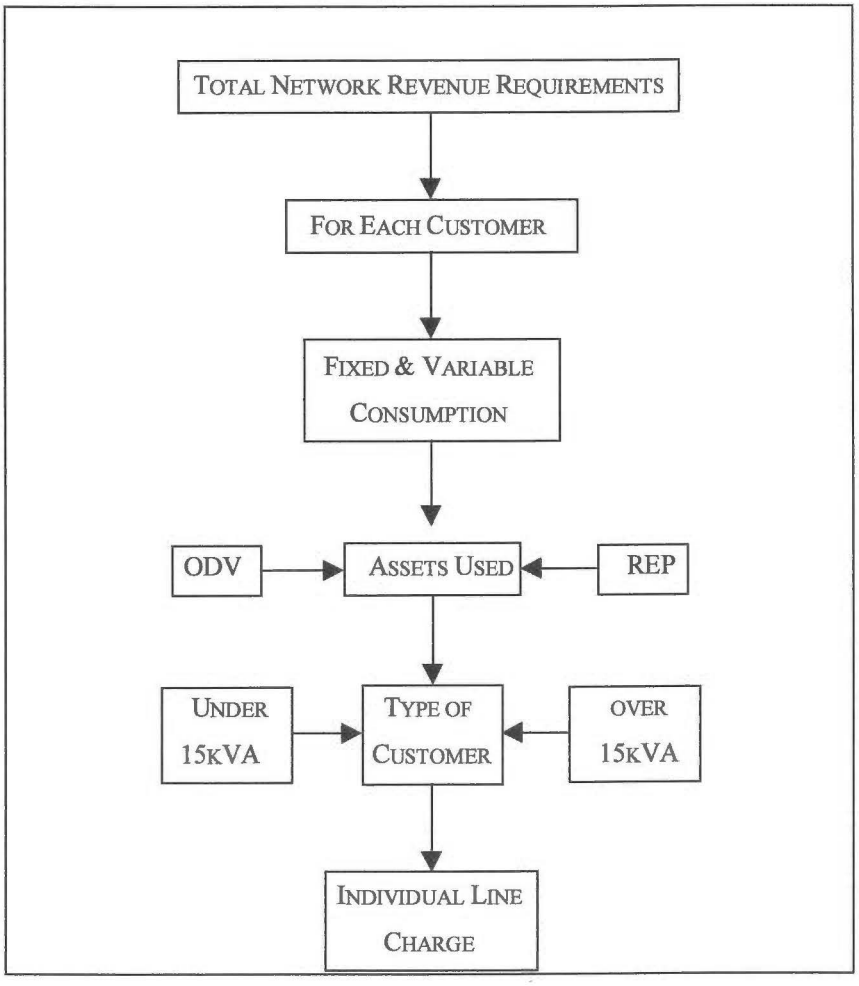

Figure 4 New Allocation Method

8) Resources Available

Alpine Energy operates four major databases. One is a general purpose accounting package and is not of concern to this project. Sections 8.1 to 8.3 present a brief description the remaining databases and the context in which they are used by this project.

The first of the two major tasks of this project was to find a method to connect these three independent databases in a way that would allow correlation and analysis of the data contained within them.

8.1 Gentrack - ENS

The Alpine Energy network is managed with the help of a computerised database called Gentrack ENS (Electrical Network System). This database forms a complete virtual representation of the physical layout and properties of the Alpine Energy HV distribution network. Because of this fact, the Gentrack ENS database was used as the core of the program, with the remaining databases being related back to it.

Access to this database was complicated by the fact that at the time this project commenced the database was held on a computer at Dunedin Electricity (in Dunedin), and accessed via the United Electricity leased line from Timaru to Dunedin. Database maintenance was performed by a third party contractor.

It became apparent that the computer system storing the database was totally incompatible for use with a PC, the system is an early Unix-variant and the operating system prevents the direct export of the actual Gentrack ENS files to a PC compatible format.

This necessitated the services of the third party contractor in order for a data extraction program to be written and run. Negotiating the contract price, conditions and gaining the necessary approvals were all completed as a part of this project.

The terminology used by the Gentrack ENS database was adopted throughout the project and in the writing of this report. It is convenient to define some of these terms now with reference to Figure 5 below.

- A site (earth site) is any physical position on the network where there is an earth rod in known conditions (ie. earth resistivity), where a piece of *equipment* may be earthed.
- Equipment covers a range of HV items in the network, including switches, fuses, links, and transformers.
- Any two *sites* may be connected by a single segment. This is simply a representation of a power line or cable.
- Installations represent the customers attached to the LV network and via a distribution transformer back to the HV network..

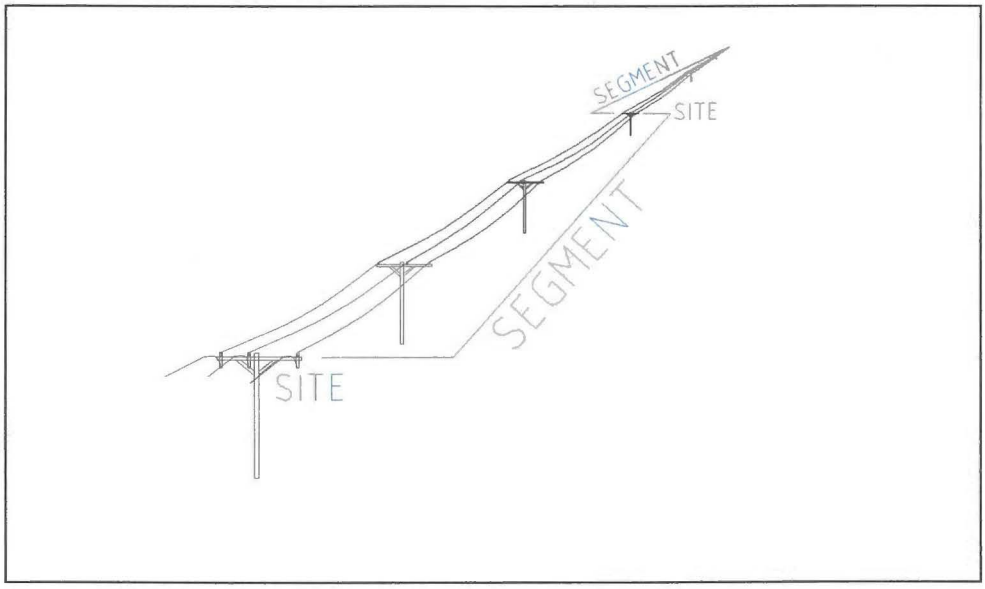

FIGURE 5 GENTRACK ENS DEFINITIONS

Every site has a unique name consisting one letter and a number. The letter represents the area the site is in (W = Waimate, M = Mackenzie Basin, L = Levels, etc..) The number is between 1 .. 999.

The name of each segment is unique and adds the prefix "S-" to the site name that feeds it. For example site F100 would feed segment S-F100. Each site can only feed one segment.

The format designed for the extraction of data from the Gentrack ENS database reduced the multi-level relational database to a flat series of inter-related text files. The structure of these files is shown below in Figure 6.

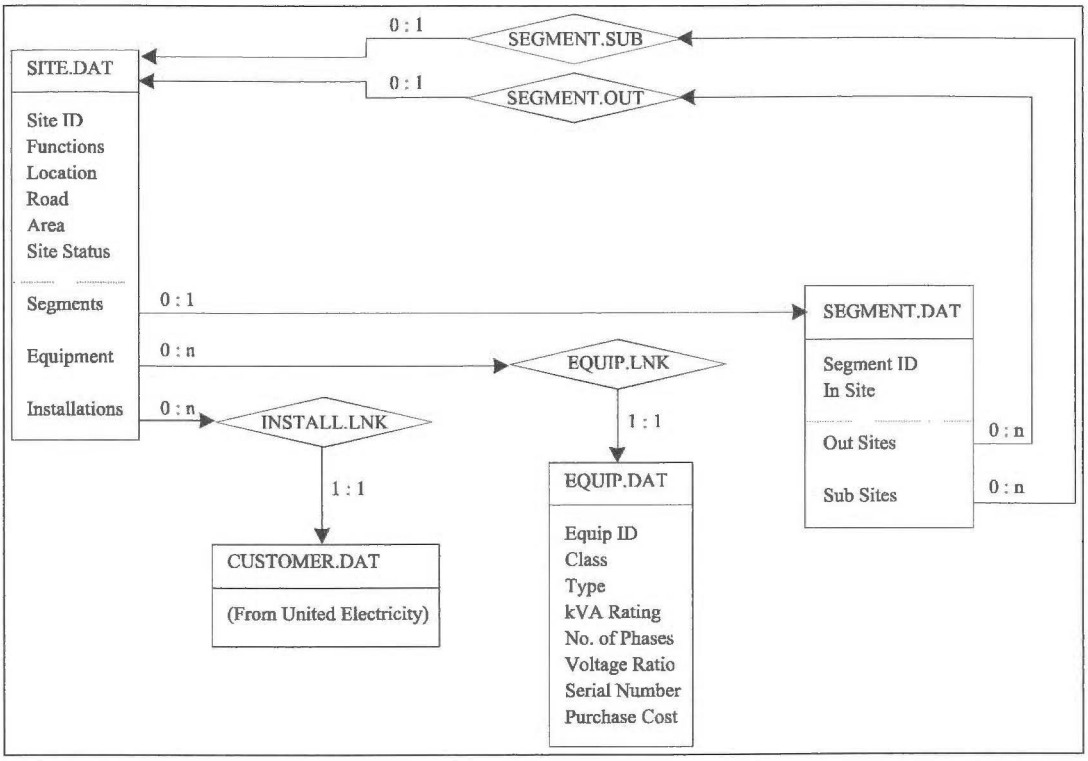

FIGURE 6 ENS DATABASE RELATIONSHIP DIAGRAM

The ENS data files can be divided into two types. Those that represent and list the attributes of a physical item (Site, Equipment, Segment) shown in boxes, and those that link them together (in diamonds).

A detailed definition of these files is contained in Appendix B2 of this report.

The relational database is not rebuilt for use with this project. Instead, the data files are used in a look-up table format to create the relationships as they are required.

8.2 UNITED ELECTRICITY CUSTOMER DATABASE

United Electricity, based in Dunedin supplies almost all[1] of the customers attached to the Alpine network. For this reason their customer database is used as the source of information for customer installations. This database is indexed by customer number, and relates to the Gentrack system by the distribution transformers common to each.

These distribution transformers are the link between the HV (33 & 11kV) and LV (415V) distribution networks. Each installation attaches to the LV network, then through a single distribution transformer to the HV network.

Access to the United database required gaining the formal approvals of both United Electricity and Alpine Energy, then generating a formal request for the data.

[1] With the exception of a few very large commercial customers.

8.3 ASSET DATABASE

A final database lists the financial values of assets[2] in the network. However, the key[3] to this database is different to that of the Gentrack database and it contains no records that would allow cross-referencing like the United data.

The asset database key changes each time the physical characteristics of the line change, (for example the type or number of conductors) which doesn't normally correspond to a segment name change. This is shown below in Figure 7.

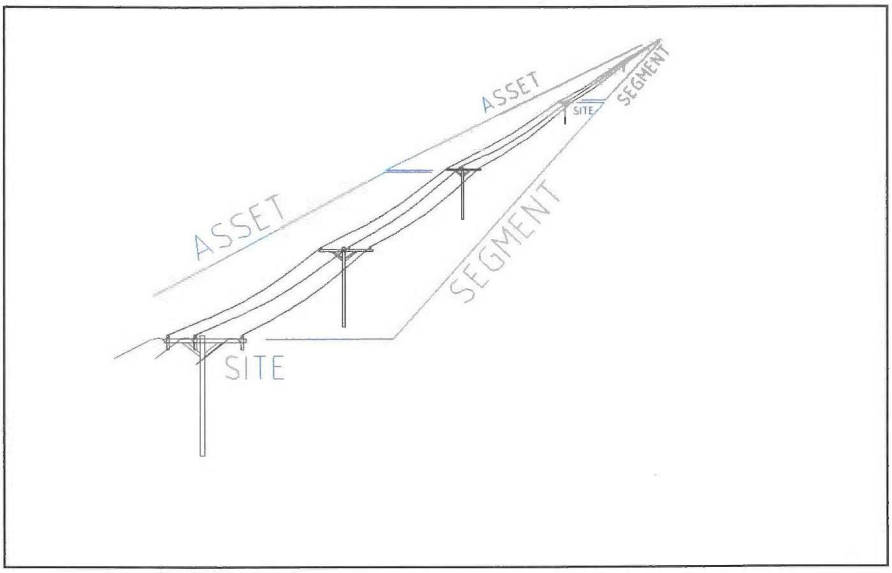

FIGURE 7 ASSET NUMBERING SYSTEM

Up until now, these factors have prevented the linking of the two databases, which would allow the correlation of physical and financial data. The only place where the keys for these databases are connected is a series of 1:20,000 maps of South Canterbury owned by Alpine Energy.

The asset database contains many different methods of valuing assets. However, only two of these are utilised in this project, they are the RC (replacement cost) and ODV (optimal depreciated value) methods. These two methods are now described.

---

[2] In this case, "Asset" refers to a section of line with exactly the same physical properties.

[3] The unique name or number that indexes a record in a database.

8.3.1 Replacement Cost (RC)

Consider a section of transmission line, assigned the asset number 12345. This line was strung twenty years ago and at that time had an expected life of forty years.

The replacement cost of this asset is the cost (today) to purchase the exact same length of line with exactly the same properties.

8.3.2 Optimal Replacement Cost (ORC)

Although not used in this project, the ORC of an asset forms the basis for ODV. In considering the above example, instead of simply replacing the line the surrounding community is examined and the question asked:

"If this line was being installed for the first time today, what type of line would be used?"

This decision could take into account such factors as expected growth (decline) in the area, environmental conditions, or the choice to replace the overhead line with underground cable.

8.3.3 Optimal Depreciated Value (ODV)

The ODV of an asset takes the optimal replacement found above and depreciates this by the current age of the existing asset. This gives a valuation of the asset as if today's optimal asset had been the one originally installed.

8.4 1:20,000 Asset Maps – A Graphical Database

A set of 93 A0 sized maps are the only source of information relating the asset database to the physical structure of the network, and thus the ENS database. These maps show the HV distribution network at a scale of 1:20,000, with the asset numbers written beside their relevant sections of line.

The different ways in which the keys to the asset and Gentrack database are defined (shown in Figure 7), preclude a direct 1:1 relationship between the two sets of key values. Therefore a third (fraction) field is created that relates how much of the asset number occurs on the relevant segment.

The first part of this project involved the creation of a database linking the asset and Gentrack databases. This was necessary to calculate the value of each segment, and thus the value of assets utilised by each installation. A graphical representation of this connection is shown below in Figure 8.

The format for this new database is: Segment ID, Asset Number, Fraction.

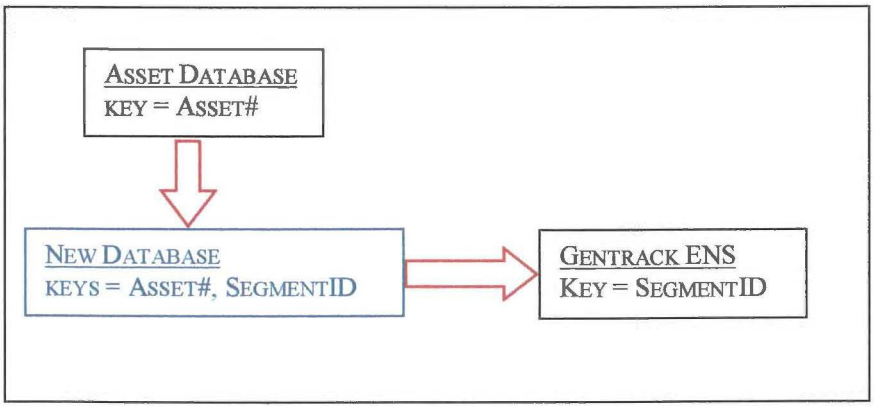

Figure 8 Linking the Asset & ENS Databases

Once a procedure for collecting this data was established, parts of the data collection and data entry tasks were contracted out.

8.5 GIS – Geographical Information Systems

Some motivation for this project stems from the possibility of Alpine Energy moving to a GIS system for network evaluation and record keeping. A GIS system would start with a computerised copy of the 1:20,000 maps of the area and superimpose on them all of the databases so far mentioned.

This allows the data from these databases to be indexed (and accessed) graphically in addition to the existing database indices.

Therefore the links between databases created by this project form an important step in the preparation for this move.

For the software created by this project to remain useful in a GIS environment the flexibility of being able to accept a significant change in the structure of input data with relatively little reprogramming was important.

9) The Computer Program

The following section outlines the basic structure and operation of the software created by this project. It is recommended that the reader considers the software user manual and data conventions situated in the appendices of this report for a more complete understanding of the software.

9.1 Development Plan – Objectives

All the data required by Alpine Energy to successfully complete this project was readily available in the format of the three databases discussed in section (7). The first major task of the project was to compile a database that links the asset and ENS data, the details of this exercise may also be found in section (7).

The second objective was to work with Alpine Energy to create a computer program that draws the information from these three databases together in a useful and flexible way. This software was titled ALP – Alpine Line-charge Program.

Motivation for this project stemmed from the fact that all of the necessary data has been available for some time, but because of the size of the network;

- 27 000 consumers,
- 6 800 sites,
- 6 600 pieces of equipment,
- 1 900 segments
- 5 250 classified assets
- and 3 500km of line;
- All spread over 3 different databases

attempting this task by hand would clearly not have been feasible. However, a customised computer program should have no problems with the *size* of data once the basic *procedures and rules* to be followed were worked out.

9.2 Abstraction of Data – Why

The task of gathering (and creating) the data required for this project was carried out at the same time as writing the software. For this reason a method known as data abstraction was employed to allow the software to be written in absence of the data it uses.

This is achieved by structuring the program in such a way as only a very small part of it actually comes into contact with the raw data. For the period that the data is unavailable, these interfaces between the program and the data are simply told to create sample data, or pointed at simple dummy data files.

The first task in writing the software was to decide on the internal data structures that would be used. These data structures can be thought of as a card-file with the same categories on each card. The only contact with the actual data is by the procedure that loads the data onto the "cards" at the start of the program. From that point on there is no contact with the "real" data.

Separating the software development from the data collection allowed half of the software to be written by the time the data was available, thus saving three to four months of project time.

A further advantage of separating the bulk of the software from the data is that the type and format of source data may be changed with relatively few changes to the software package.

9.3 TEXT DATA FILES – WHY

Comma separated value (CSV) was the data format chosen for use with this package. The reasons behind this are simple, but worth mentioning:

1. CSV data is a universal standard.

2. Although not the most efficient use of space, time is saved by not having to decode compressed data from disk while the program is running.

3. Plain text data allows the results of the program to be checked manually. An important consideration if the results are to be capable of being audited.

4. With the data files in a clear text format, there exists the ability to change the data within the text file by hand.

All of the input data files and the files containing the results of a run are in CSV format.

9.4 THE SOFTWARE

The following sections look at the basic properties of the software created for this project. All software was written by the author, with some debugging and checking performed by Yvonne Simmons (MEM 1997) and Gerard van Soest (3rd Pro Elec 1997). A more complete description of the software is contained in Appendix A: Software User Manual at the end of this report.

9.4.1 Platform and Interface

ALP was created to run on a Microsoft Windows[4] based 32-bit operating system. This requires Windows 95, Windows NT or higher.

The primary (day-to-day) interface to the program is graphical, although allowance has been made to alter program and data parameters through a Windows 3.1 style .ini file. The choice to use a text-based ini file in favour of the system registry was made to allow ease of change by the end user.

A detailed description of the user interface is contained within the program user manual, included as Appendix A to this report.

The advantages of using a 32 bit operating system are:

- The program will perform better on a machine that has the minimum specifications for running a 32 bit operating system.
- All program data is loaded from the CSV files at the start of the program. A 32 bit system provides a better memory structure for handling this large amount of data.

With reference to the second point; the first copy of the software operated directly on the data files, which was found to be very time consuming. The current version incorporates a MasterList data structure which replaces the need for direct file access.

[4] Microsoft, Windows, Windows 3.1,Windows 95 and Windows NT are all registered trade marks

9.4.2 Theory of Operation

One requirement of this project was that any solutions had to be transparent in their operation. This is stipulated by the legislation governing the electricity market. It should therefore be possible to adequately describe the full operation of the software in plain English.

The goal of the software is to assign the cost of running the Alpine Energy network to consumers using a method that considers the dollar value of assets used by each consumer.

This is a variation on the ROI[5] theory of cost recovery. Under ROI, the business totals the capital investment employed in generating income and requires a percentage of this value as revenue.

The method employed by the ALP software starts with a target revenue, then divides this up based on ROI theory, but using a target income to be generated rather than a fixed percentage of the dollar value of assets employed.

9.4.3 Intermediate Data Files

Because of the scale of the computations required, two intermediate data files were created to speed up the actual calculation runs. These intermediate files need only be recalculated when the ENS or Customer databases are changed.

These two files, taking up to an hour in total to assemble allow program execution a factor of 10 times faster than before they were calculated.

The task performed in the creation of these intermediate data files is the linking of the asset and ENS databases. Due to the differences in keys for these databases this is a large task and performing it once when the data is changed is more efficient than recalculating it each program run.

[5] Return on Investment.

9.4.3.1 Tracing Dependants

This task involves looking at each segment in the network and calculating the total load[6] on it. This load includes all installations off the chosen segment and installations off all segments downstream of the chosen segment. An example of this is shown below in the switching diagram of Figure 9, where blue shows the lines dependant on the chosen segment. Note that the site feeding the chosen segment is highlighted as well.

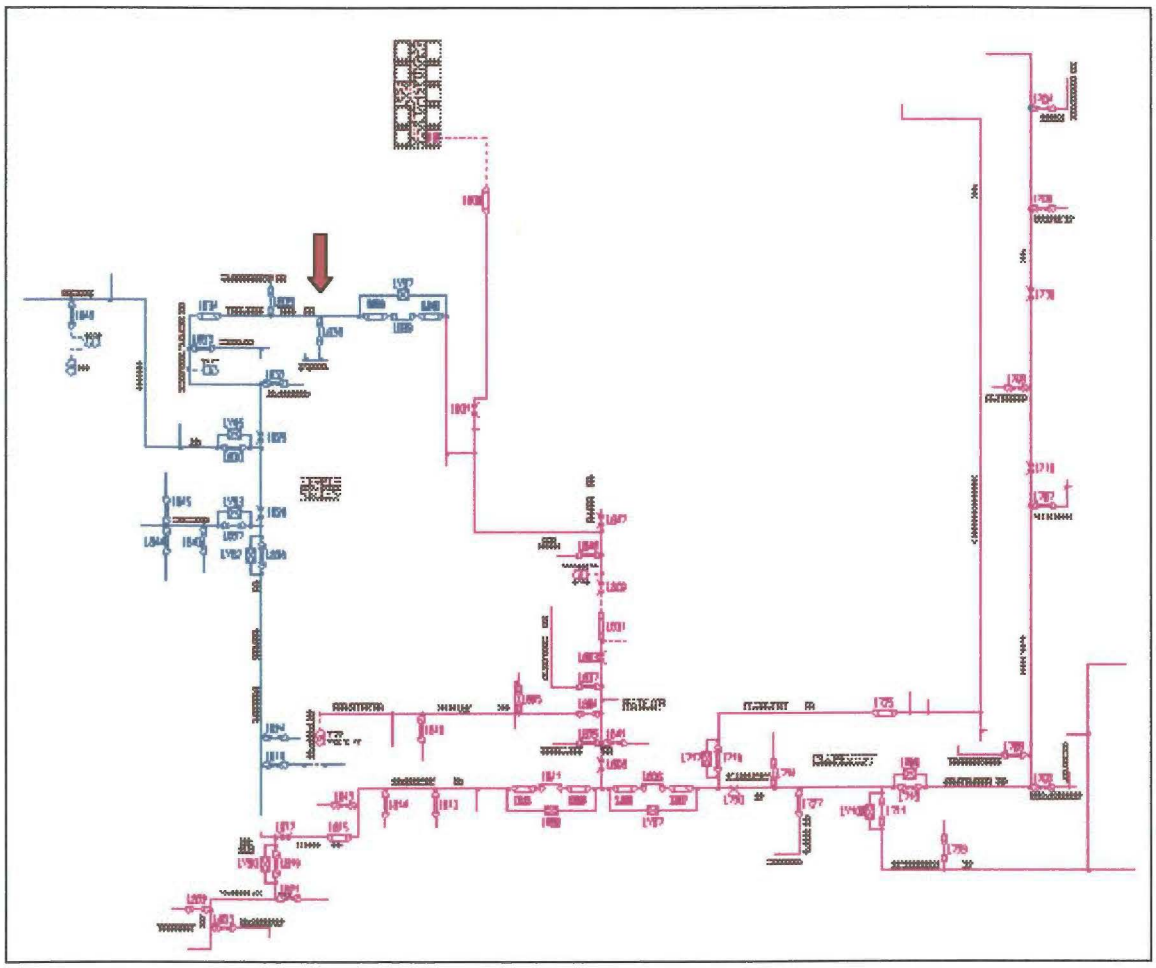

FIGURE 9 TRACING DEPENDANTS

A second algorithm uses the asset database to calculate the value of the chosen segment.

[6] KWhr and kVA

With knowledge of the loading on and value of each segment it is possible to divide the dollar value by the different load measures to get $/kVA and $/kWhr for any installation downstream of this segment.

For example, a segment worth $100,000 with 1500 kVA of installation capacity downstream of it would be valued at $66.67 per kVA. As all residential households are considered to have a capacity of 15kVA, a household would be apportioned $1,000 of this particular segment.

9.4.3.2 Tracing Source

Once the dollar per load measure is known for each segment, the reciprocal task is performed. Starting with distribution transformers, the load on each transformer is calculated. This load value is used to determine the portion of the network utilised by this transformer by tracing back through segments to its point of supply. At each step (segment) the dollars per load measure (for the segment) are multiplied by the load (at the transformer) and added to the running total.

In this way a second data file is created listing the dollars per load measure for each distribution transformer in the network.

Therefore when a calculation run is performed, each installation need only be traced back to the distribution transformer it connects to, and the transformer looked up in the intermediate file to find the dollar value of assets utilised by this installation.

9.5 SOFTWARE METHODOLOGY

The software package for this project was written using a highly functional, highly cohesive method. This means that there are many "small" procedures divided into many modules, each with a single well-defined task. Advantages of this method include increased ease of understanding and maintaining the code and increased flexibility of the software.

9.5.1 Module Hierarchy

The breakdown of the program code into modules is shown below in Figure 10. Each module contains a series of functions related to its topic.

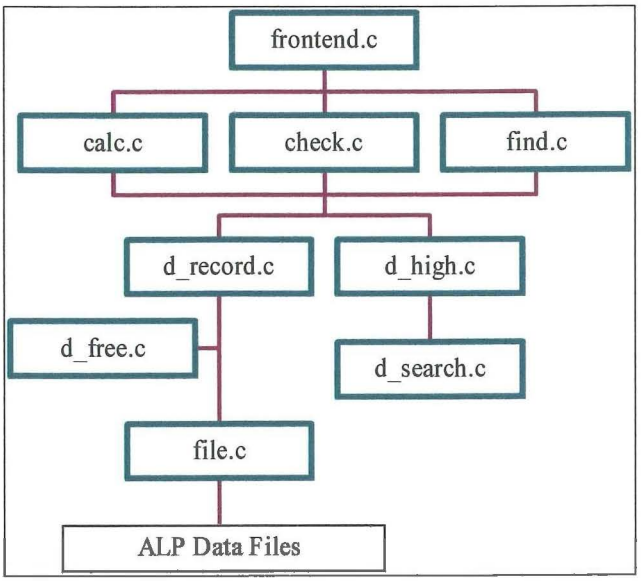

FIGURE 10 SOFTWARE MODULE HIRERACHY

The following section contains definitions of each of these modules. The header files that relate to these modules are contained as appendix C to this report.

It can be seen that the single point of contact with the user (frontend.c) and single point of contact with the data (file.c) shown in Figure 10 make the software interfaces very versatile and easily changed.

9.5.2 Definitions of modules

The total length of the code is in excess of 8,000 lines. This is made up of 7,000 lines of "program" code in the form of .c source files and some 1,000 lines of "header" code (.h files), containing function prototypes, data types, and definitions.

Frontend.c This modules controls the windows interface of the program.

Find.c Selections from the find screen are forwarded to this module for processing.

File.c Performs all file I/O operations. Including loading the data files at the start of the program. This is one of the two modules that would require attention if the format of input data changed away from CSV.

Check.c Data checking operations from the main menu bar of the program are processed by this module

Calc.c Performs actual line charge calculations, calls the data modules for most of the functions.

Data All of the data manipulation functions are grouped into a series of four data (or d_) files.

The relative sizes of the above modules are shown in Figure 11, followed by a description of the d_ files.

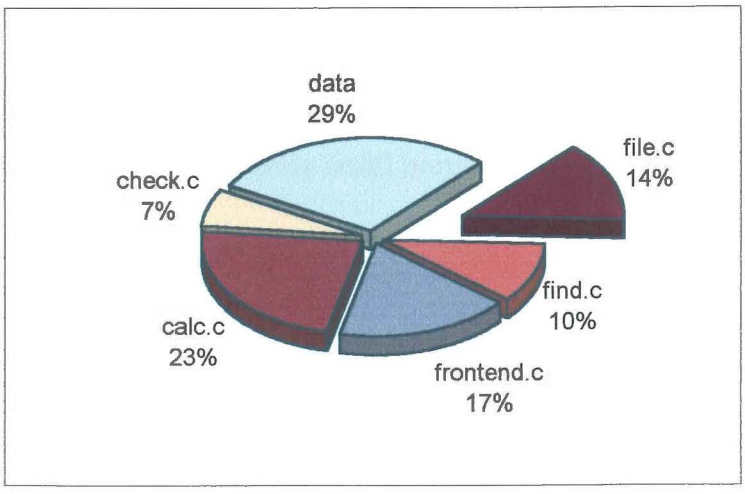

FIGURE 11 RELATIVE SIZES OF PROGRAM CODE

d_record.c Deals with the loading of data from file (by calling procedures in the file.c module) into the data types used by the software.

d_free.c Unloads any of the above data types from memory.

d_search.c Performs simple search operations in the data. These are generally one-step searches. For example, given a site – what segment does it feed, or given a distribution transformer – what segment does it attach to?

d_high.c Calls search.c functions to perform high-level search operations. For example, what segments feed through this segment, what is the loading of this segment, which installations are fed from this list of segments?

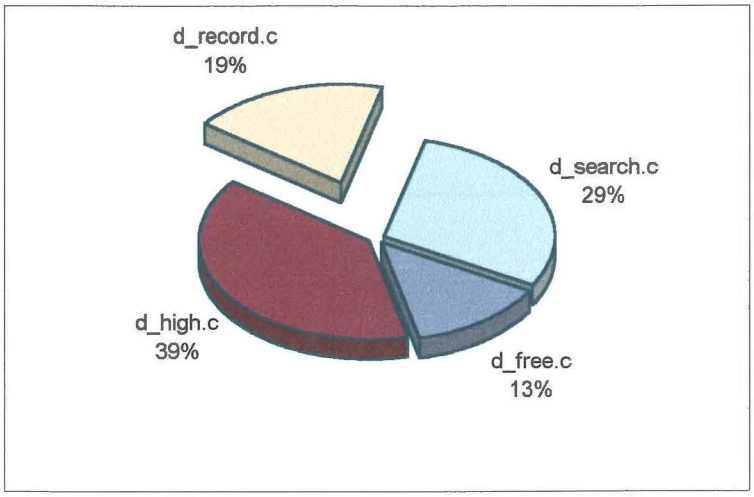

FIGURE 12 RELATIVE SIZES OF DATA MODULES

Therefore, a change in the input data file format will only require changes to the file.c and d_record.c procedures, a relatively small portion of the software package.

9.5.3 Internal Database Handling

The relational structure of the Gentrack ENS database is not rebuilt within this software package. Instead the text files are used as a series of (flat) look-up tables that are traversed by a series of very simple query functions.

High-level search procedures build on these simple query functions to allow complex queries across the four databases. The fields that link the databases together are shown in red on Figure 13 below.

The choice not to rebuild a fully relational data structure was made because of the relatively few rules governing the Gentrack ENS database. A segment may have *n* sub-sites (distribution transformers) and *n* out-sites fed from it, and each sub-site can feed *n* consumers. By retaining the flat structure of the data files, errors due to the incorrect re-building of the ENS database were avoided. It also makes the code faster to load and easier to follow.

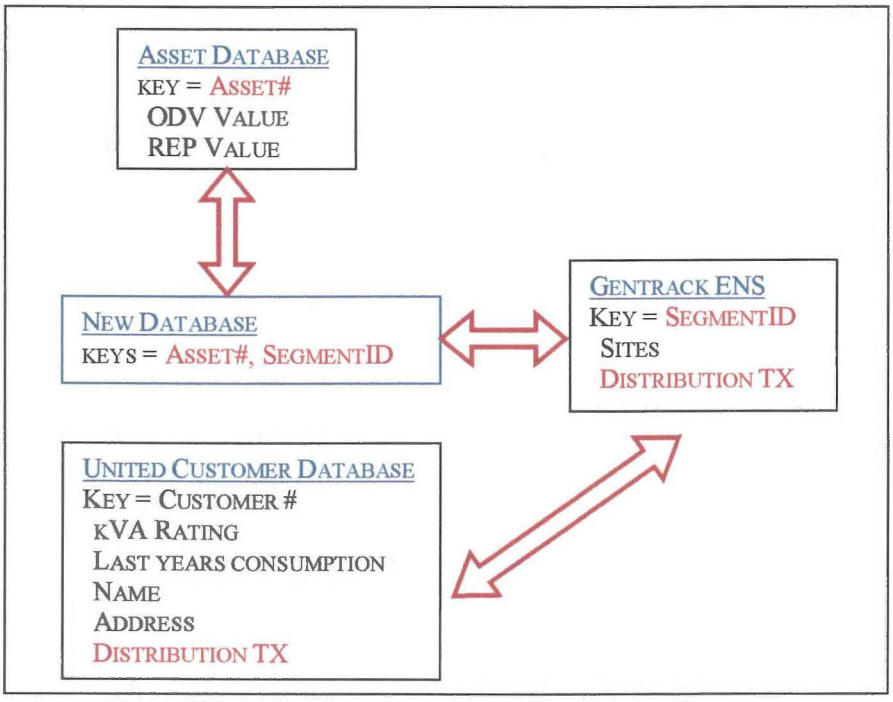

FIGURE 13 DATABASE LINKS

10) PROGRAM EVALUATION

10.1 STRENGTHS OF THE PROGRAM

10.1.1 Abstraction

Removing the bulk of the software from the format of the input data allows for a program where relatively little software rework is required to accommodate any changes in input data.

10.1.2 Flexibility

There are a considerable number of options available for calculating line charges. Including the base on which they are calculated (asset-based or per-customer), the valuation of assets (ODV or REP), the resolution (POS/Feeder, Segment, TXD) and the percentage ratio of fixed to variable charges.

10.1.3 Y2k Compliance

This software, to the best of the author's knowledge is year 2000 compliant. There are no direct references to dates within the package, other than a routine that extracts the computers BIOS date and time to put on data files.

However, it was observed that the current format of the asset database will make it susceptible to the Y2k bug. This stems from the fact that the "date of installation" column is only two digits wide. It has been recommended that this be adjusted to a four digit number in the near future.

10.2 Known Weaknesses

10.2.1 GIGO

It is pretty obvious that the accuracy of this program is only as good as the data put into it. Unless care is taken to maintain the databases and their inter-relationships, the validity of the results will decline over time.

10.2.2 Line Extensions

Imagine a new house, built just past the end of the nearest LV power line. The owner approaches Alpine Energy for a connection to the network. Alpine agrees to extend the line, provided the new house owner pays a portion of the capital cost of the extension. This effectively purchases the new house owner the right to pay standard line charges.

However, only the total cost of the line is recorded in the asset database while the knowledge of who paid for it is lost.

Therefore this program will allocate the entire cost of the new asset to the customer, not recognising that the customer has already paid for a portion of it and should not be charged for that last section of line.

10.2.3 Changes in POS

In creating the software, the names of the various points of supply (POS) around the network (as defined in Gentrack) are integral to the control flow within the software.

The integral nature of the POS in the software allowed a known point of reference for the start of calculations and eliminated almost all unknown elements from the variables input into the program. Thus eliminating the need for complicated input and input validating procedures.

Provision was made to change the names and definitions of the POS, and a POS was changed during the development of the software with the introduction and commissioning of the new Clandeboye sub-station. The procedure taken to add this POS was documented for future reference and included as appendix D to this report.

11) RESULTS

11.1 ANALYSIS & TRENDS

Analysis of the results generated by this project will take considerable time. The permutations of alternative line charge strategies offered by the ALP software run into the tens of thousands.

Alpine Energy now has a powerful tool that will allow them to investigate the cost recovery of their network in an asset-utilisation and per-customer setting.

The software is capable of delivering many different line charge strategies that are fair from an asset-utilisation viewpoint. However, this does not necessarily produce a method that is acceptable for implementation.

The basic trend observed for consumers under a purely asset-based line charge method was that asset-based line charges were distributed relatively closely around the current line charges. This is shown in Figure 14 below, with the red arrow showing the level of the current line charge.

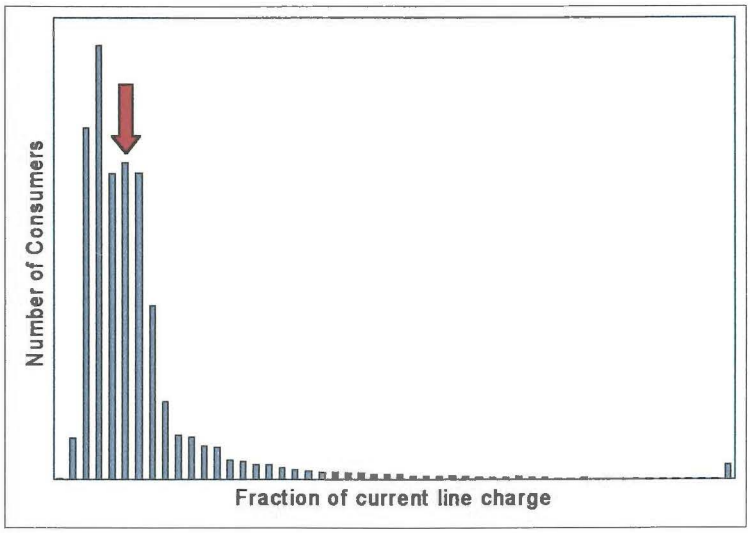

FIGURE 14 COMPARISON OF LINE CHARGES

Figure 14 is a histogram automatically generated by Microsoft Excel 97 and shows what a powerful tool this is for graphically analysing the 27,000 lines of results from a full scale program run.

The histogram of Figure 14 can be interpreted as showing that a large proportion of consumers would pay slightly less under an asset-based scheme (those to the left of the arrow), and are therefore subsidising a very small proportion of consumers (to the far right of the arrow).

This subsidy makes intuitive sense as 10,000 consumers subsidising 100 consumers by a fraction of a cent per kWhr will allow the line charges of these 100 to be reduced substantially.

For example if 10,000 consumers each pay an extra dollar per year in line charges (0.01 cents per kWhr[7]), then the line charges of the 100 being subsidised may reduce by $100 per annum.

There are many scenarios such as this one around the Alpine energy network which can be generated and analysed at will with the use of ALP and Excel.

NOTE: In order for this report to remain freely available a more detailed analysis of the results has been omitted. More information *may* be made available by contacting Alpine Energy, PO Box 530, Timaru.

11.2 IMPLEMENTATION

The theoretical maximum line charge is 40 cents per kilo-watt hour. At this point it is more cost efficient for the consumer to invest in their own energy generation, rather than using the Alpine Energy network for electrical supply. Comparison to the current line charges of ~5 cents per kWhr show that this maximum is a long way away.

It is recognised that a line charge even approaching this maximum amount will lead to a considerable back-lash from consumers. A blindly applied asset-based line charge is not suitable for implementation.

[7] The standard consumption for a household for a year is assumed to be 8,500 kWhr

However, the ALP software may be used to produce line charges using the current method, but on a POS basis, rather than the network-wide basis currently implemented.

This will allow Alpine Energy to recover line charges from the network in a way that is proportional to the assets attached to each POS, but averaged over all consumers on that POS to reduce the impact of changes.

Another use for the software will be to determine at which points on the network asset-based cost recovery fails under the current method. As the project objectives speculated, the current line charge methods cover the costs of assets utilised up until a certain distance from the point of supply. The very small numbers of customers on the far right of Figure 14 are, as a rule in isolated rural areas on the end of a ling transmission line.

Alternatively, the software may be used to identify particularly uneconomic sections of the network that are recovering sufficiently little revenue that alternative energy strategies may be considered. These could include remote generation, alternative line configurations or a myriad of others.

In this case, the implementation of demand side management may become economical in rural areas if the line (or energy) charges increase. This is where the home owner installs equipment capable of managing energy around the home. For example, running dishwashers, washing machines and other high-usage appliances at night when electricity is cheaper, or lowering the energy requirements of the house by increasing efficiency. Although this will have no impact on fixed line charges, the variable component will reduce as consumption reduces.

12) FURTHER DEVELOPMENT

This project has created a software package capable of spanning the Alpine Energy electrical distribution network and assigning the revenue recovery of the network to individual users.

The data files and links between databases created by this project make the proposed move to a GIS system for the Alpine Energy network more feasible now that the asset database may be linked to the physical network.

There remains the possibility that if a new GIS system is installed at Alpine Energy, it will be able to perform the tasks of this software internally. However, the provision has been made for the integration of the ALP software package into a GIS environment.

It should be noted that the methods used in creating this software package make it incredibly adaptable. The input data may be changed and the hierarchical programming methods mean that even if a different type of network is used, the high-level search and calculation procedures will remain valid.

Therefore this software may be adapted to other utilities, or asset-management situations. For example:

- Other electrical energy distributors,
- Water or sewage services,
- Telephone (Ground and Cellular based) systems,
- Public transport networks,
- Road, rail, sea and air transport of goods or people,
- Pricing of services based on assets used and distances travelled,

13) REFERENCES

[1] "Electricity Information Disclosure Compendium", The Ministry of Commerce, June 1994.

[2] "New Zealand Electricity Directory 1996", Chameleon Enterprises Limited, 1996. ENQUIRIES TO: C.Wylie, PO Box 10374, Wellington

[4] Flemming, D.R, "Methodology for the derivation of line business charges", 25 March 1992. Prepared for King Country Energy.

[3] Acts and inquiries relating to the deregulated energy market:

- The Stanton Commission, 1959
- The Electricity Distribution Commission, 1971
- Electricity Taskforce, February 1988, (Report September 1989)
- The Electricity Distribution Reform Unit, 1990, (Report March 1991)
- The Energy Companies Act, 1992
- The Electricity Act, Gas Act, Energy Companies Amendment Act, all 1992

14) APPENDICES

A. A.L.P. User manual

B. A.L.P. Data Conventions and Procedures

C. Header files for software

D. Change of POS procedure.

E. Glossary of Terms

F. Index of Terms

DERIVATION OF LINE CHARGE ALGORITHMS

FOR

ALPINE ENERGY LIMITED

A.L.P. – ALPINE LINE-CHARGE PROGRAM.

SOFTWARE USER MANUAL

By

CHRISTOPHER SAVAGE

UNIVERSITY OF CANTERBURY

DECEMBER, 1997

1) A.L.P. Alpine Line Charge Program

This help file contains the complete on-line version of the documentation provided with this program.

2) About A.L.P.

This program was written by Chris Savage in 1997 as a project for the Masters of Engineering Management course at the University of Canterbury.

The program is designed to assign line charges to customers of Alpine Energy in a way that is consistent with Department of Commerce guidelines. The line charges are calculated based on the asset use of each customer.

Software for the main package has been written using Borland C++ version 4.5 at the University of Canterbury. This help file was compiled using a shareware program HelpScribble version 3.0.3.

This software is provided "as is". In no event shall I, the author, be liable for any consequential, special, incidental or indirect damages of any kind arising out of the delivery, performance or use of this software. This software has been written with great care but I do not warrant that the software is completely error free.

Source code for the main application file has been supplied with this program. However, any changes to the original code are not my responsibility.

3) Directory Structure

A.L.P. assumes a specific set of data files are available in pre-defined directories.
The program itself (as a whole) may be placed anywhere, in any directories - but there must be a specific set of sub-directories under the directory containing the file ALP.EXE
This structure is:
("." indicates the root directory that the *alp.exe* file is located in)

.*alp.exe*	- The main executable file for the program that you run.
.*alp.ini*	- A text file that contains initialisation data for the program as well as a summary of details for each feeder.
.*bwcc.dll*	- Used by the program to perform windows functions
.*cw3215.dll*	- Used by the program to perform windows functions
.\A2S*a2s.csv*	- This text file is the database that relates the asset database to the Gentrack ENS database and contains three columns. The segment name, an asset number on that segment, and the portion of the asset on that segment.
.\Alpine*assets.csv*	- This is a comma-separated-value dump of the Alpine asset register. It contains the dollar values of each asset number.
.\ENS*equip.dat*	- Records relating to equipment on a site. Key = equipmentID
.\ENS*equip.lnk*	- Relates equipment (ID) to site.
.\ENS*install.lnk*	- Relates Installation (Number) to a distribution transformer.
.\ENS*segment.dat*	- Relates segment (ID) to the site feeding it (inSite).
.\ENS*segment.out*	- Relates segment (ID) to the sites it feeds (outSite).
.\ENS*segment.sub*	- Relates segment (ID) to the distribution transformers it feeds (subSite)
.\ENS*site.dat*	- Records relating to (earth) sites. Key = siteID.
.\HELP*alpine.hlp*	- This help file.
.\OUTPUT\	- This is the directory where all program run data files are placed. Also the two exception report files *S2Aex.txt* and *ADex.txt*, and the error log *error.txt*.
.\progDATA*LVtxd.dat*	- Program created records relating to TXDs. Key = TXD.
.\progDATA*segKVA.dat*	- Program created records relating to segments. Key = segmentID.
.\SOURCE\	- The complete C source code for the A.L.P application (excluding the help file).
.\UNITED*ael.csv*	- The customer data file from United in comma separated value format.
.\UNITED*customer.dat*	- The processed United database with "pretty" aspects removed.

4) Main Screen

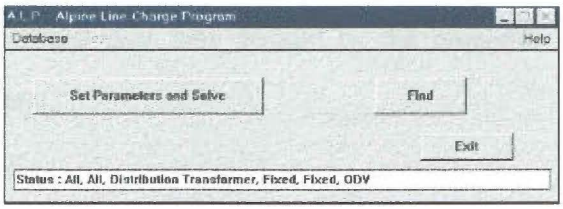

FIGURE 15 MAIN SCREEN

The main screen of this package consists of a menu bar, three buttons and a status bar.

Menus: These menus provide access to lesser-used functions of the program and this help file

Buttons: The buttons provide access to the selection of the part of the network to be examined through the settings and solve screen, which is followed by a series of screens that allow the program parameters to be set. The Find Screen allows access to the database to perform searches for individual entities. The third button allows you to exit the program.

Status Bar: The horizontal, white status bar along the bottom of the main screen displays an idea of what is happening. The current status lists the type of asset database valuation currently selected (ODV or REP) and information on the 980/TOU customers. (IN or OUT)

4.1 Menus

The two menus available are Database and Help.

The help menu simply allows access to this help file.

The Database menu accesses a series of functions:

- Rebuild Database
- Check Data Files
- Format United Data
- Link Tekapo Substations
- Load non-United Installations

The greyed out "test" menu is for use during any future development of the program. It is unavailable during normal program operation.

4.1.1 Rebuild Database Menu Item

This menu refreshes the two data files created by the program, found in the ProgData directory.
This function is used when:

1. New Gentrack data, or
2. New United data, or
3. New Asset Data, or
4. New Segment-to-Asset Data arrives.

or,

5. If the Customer data is changed to include/exclude 980/TOU customers, or
6. If new sites, segments or customers are connected via the menu functions.

Any (or all) of these four new data files types must be placed in their correct directories , then the program started and the [Rebuild Database] option run from the Database menu.
This function creates two new summary files for the program to use. These files may take up to an hour to create (depending on computer, network, etc.) but reduce the running time of the program by a factor of at least 10.

4.1.2 Check Data Files Menu Item

This function creates a series of output files listing the following inconsistencies in the databases:

1. Segment-to-asset file. Checks the S2A data file against the site-to-segment file and reports exceptions. That is, segments that are listed in one, but not the other.
2. Asset Allocation. Checks the S2A data file against the asset data file, reports exceptions and inconsistencies. ie, asset numbers that are listed in one, but not the other.
3. Asset Allocation. Secondly, checks the S2A file and reports asset numbers that are allocated more than 1.0 times or less than 0.95 times.

4.1.3 Format United Data Menu Item

When the customer database arrives from United it's called *ael.csv* and most of the numbers in it (for kW, kWhr, etc..) are in a "pretty" format with commas and speech marks -- (ie "1,234,567").
This function removes these and saves the new file as *customer.dat*.
This function only needs to be run when there is a new *ael.csv* file. If the data file arrives from United with a different name, it must be renamed before use.
The original *ael.csv* file is not altered, and is NOT used by the program during calculations.

However.
This function also allows you to set the status of 980/TOU customers. By running this function you can include or exclude 980 customers from the database.
NOTE: If installations have been added via the Load non-United Installations option, they will be removed by this function. They must be re-loaded.

4.1.4 Link Tekapo Substations Menu Item

This option loads information found in the alp.ini file into the Gentrack ENS data used by the program.

The primary purpose is to link the three substations found in the Mackenzie basin together, but it may be used to add or adjust the Gentrack data used by the program.

In the ini file:

1. The TOTAL number of lines option MUST BE CORRECT! The program will look for this many lines to add. If there are more, they will be missed, less will result in corrupting the database.

2. The database is checked before each new line is added. If it is already in the database, the addition operation is cancelled for this line. This means that performing the operation twice or more times in a row will not result in corrupted data.

4.1.5 Load non-United Installations Menu Item

The provision to load non-United installations into the customer database has been made.

The format for adding data in the ini file is recorded there. This format is:

Install(#)=accountNo,CSNO,,,TXDsite,,kVArating,,,,,summerKWhr,winterKWhr

Should use field numbers 1,2,5,7,13,14 - rest are blank

where (#) is the next available number in the list below

Remember to update the numInstalls field.

5) Settings Screen

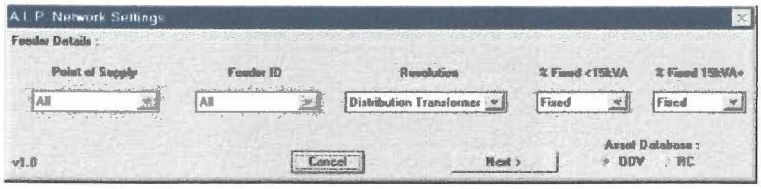

FIGURE 16 SETTINGS SCREEN

The settings screen allows you to choose the physical part of the network you wish to perform a calculation run on.

Point of Supply : Sets the sub-station that you want to calculate from the drop-down list.

FeederID : Which (or all) feeders to calculate from this Point of Supply

Resolution : How do you want the costs to be split? Everyone on the same <TXD, Feeder, POS, or Everyone> pays the same amount *per kVA and kWhr.*

%fixed <15kVA : What portion of the line charge is to be fixed (calculated per kVA) for customers using 15kVA or less connections.

%fixed >15kVA : What portion of the line charge is to be fixed (calculated per kVA) for customers using 15kVA or larger connections.

Note : The variable (not fixed) portion is calculated as a cents per kWhr - ie. is is usage dependant. Last years kWhr figures from the United database are used to calculate this.

Asset Database : To allocate the costs based on the Optimal Depreciated Value (ODV) or Replacement Cost (RC) of assets used.

Pressing [OK] will forward you to the Continue Screen. The choices you selected will be displayed in the status bar.

6) Continue Screen

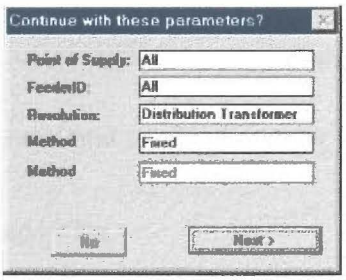

FIGURE 17 CONTINUE SCREEN

This screen simply allows you to check the parameters that the calculation run will use. You cannot change the parameters here, you must [cancel] the run and return to the settings screen.

Upon pushing [next >] you will progress to the file name screen.

7) File Name Screen

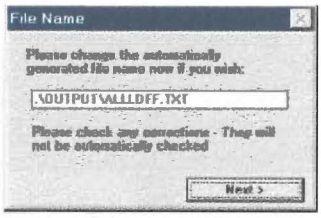

FIGURE 18 FILE NAME SCREEN

This screen allows you to change the file name automatically allocated to this calculation run.

The name you enter will not be checked. This means that long file names can be entered. However the program will also happily accept illegal file names as well.

Pushing [next >] brings up the Dollar Allocation Screen..

8) Dollars Screen

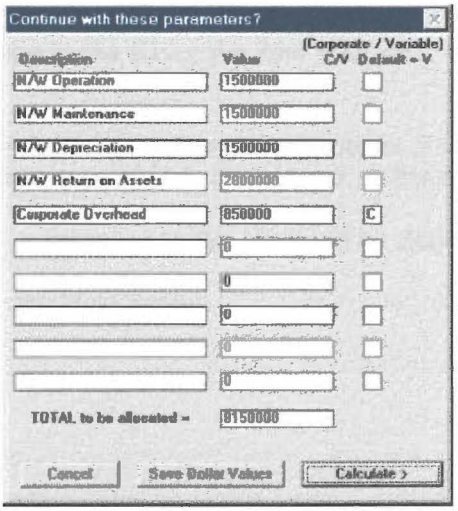

FIGURE 19 DOLLARS SCREEN

This screen allows the user to check and modify the dollar allocations to be recovered from the network.

This information is stored in the alp.ini file.

It is assumed that all costs are recovered on an asset-use based allocation. If a particular cost is to be allocated equally across all consumers the "Corporate" flag is set on the end of the line.

When the [calculate >] button is pressed, the program will return to the main screen and report to the status bar until the run is finished.

While a calculation run is occurring, you can not use any of the buttons or menus on the main screen.

9) Find Screen

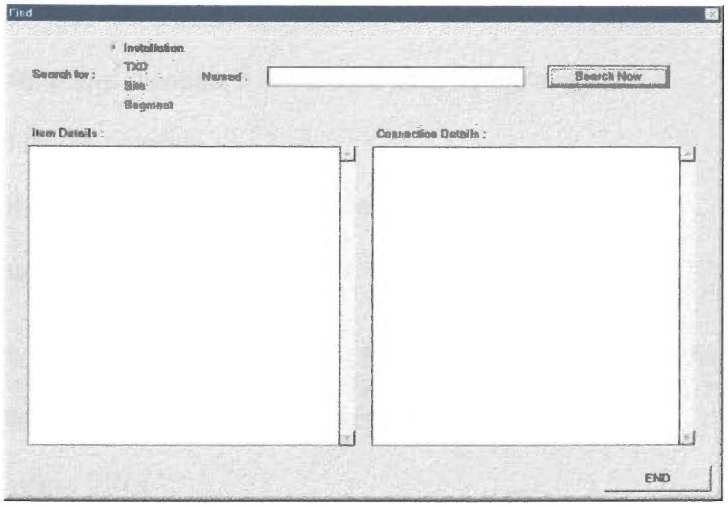

FIGURE 20 FIND SCREEN

This screen allows you to search the database for any of the five types of item listed with check boxes.

To search for something, type in the ID for the item you want, then select the type of item from the buletted list and hit the "search now" button.

If the search is successful, information relating to this object is displayed in the left window, and information relating to its connection into the network on the right window.

If the search is unsuccessful, a message to this effect will be displayed in the left window.

Pushing [end] returns you to the Main Screen.

10) Error Messages

If a database error occurs during program execution, a standard error message will appear. It lists the module the error occurred in and the task being attempted.

10.1 Error opening file

Check that the directory that this file should be in exists.

10.1 Null pointer from *x*

This means that a link in the database is expected, but not present. The name of the module *x* will indicate which data file is incorrect. ie. null pointer returned from Site2Seg would indicate that an expected line is missing or incorrect in the segment.dat file that links sites to segments.

** At the date of printing, the procedure to shut the program down after error message is acknowledged is inoperative. Manually shut down program (ctrl+alt+del, then [end task]) and restart.

11) Output Files

Three types of data files are created by A.L.P.

1. Data files reporting any exceptions in the data used by the program

2. Data files used by the program

3. Output files with solutions from program runs.

11.1 Program Files

A.L.P creates two data files that are network & asset summary files for the program to use. These files may take up to an hour to create (depending on computer, network, etc.) but reduce the running time of the program by a factor of at least 10.

11.2 Solution Files

Performing a calculation run of A.L.P. will result in the generation of a report file. The file is assigned a default name, but the user can provide another name in the File Name Screen, before the calculation run.

The default file name is created in the following way:

The first (2) letters of the POS name
+ The last (2) letters of the Feeder name
+ The first (1) letter of the Resolution name
+ The first (1) letter of the method for U15
+ The first (1) letter of the method for P15
+ .txt on the end.

11.3 Data Files

A.L.P uses a relational database to create a virtual representation of the South Canterbury electricity distribution network.

The physical connectivity of the network comes from data files extracted from the Gentrack program operated by Alpine.

The dollar values of these assets are stored in a second (Excel-based) file by Alpine.

A third database relates the first two to each other as they use different keys.

This program is based on text data files stored in a comma separated value format (csv). This is neither the fastest, or most space-efficient way of performing this task, but is does allow manual calculations of results directly from the source data.

Appendix B: A.L.P. Data Conventions and Procedures

All data files used with this program are in comma-separated-value (csv) format.

The procedures that load the csv files into the data structures used by the program RELY on certain data being in each column of the data files. The NAMES of these files are also important.

This appendix sets out the format for the data files used by the program. If the data files do not conform to these standards the program will not work properly (if at all).

In each case the primary key is double underlined. Secondary keys (if present) are single underlined. Note: the key is the field in the database that contains a series of unique entries.

It is recommended that any new data is checked before being integrated into the program and that old data is backed up first.

A definition of the field names and the type of input expected appears after each file definition. The format for these is : Field name, Description, format.

Any important conventions that are not immediately obvious are included as violet text in a grey shaded box. As shown below and above.

The maximum length for any single CSV record (line) is 200 characters.

Gentrack ENS files:

These files are kept in the .\ENS\ directory of the program as outlined in the Software user manual of Appendix A. The field names specified are the standard ENS ones used for data extraction.

The contact for these files is:

Grant Smith

ACE Computer Consultants Limited

17 Arthur St

Invercargill

Phone: 03-218-3322

e-mail: ace.ccl@southnet.co.nz

Grant wrote the original extraction program to create the data files listed in this section directly from the Gentrack ENS database.

EQUIP.DAT Records relating to equipment.

```
EquipID,Class,Type,kVA,Ph,VoltRatio,SerialNo,PurCost
```

EquipId	Unique identification tag.	30 characters
Class	Transformer / Non-transformer	5 characters
Type	of transformer	5 characters
KVA	rating of equipment	integer number
Ph	number of phases	integer number
VoltRatio	of transformer	30 characters
SerialNo	on equipment	30 characters
PurCost	purchase cost of equipment	floating point number

EQUIP.LNK Relates equipment to site.

```
EquipID,Site
```

EquipId	Unique identification tag	30 characters
Site	Name of earth site attached to	30 characters

INSTALL.LNK Relates Installation (Number) to a distribution transformer.

```
Install No,TxSiteID
```

Install No	Customer number	Long integer
TxSiteID	TXD that install connects to	30 characters

* This data is not used by the program. Instead the Trans No column of the United customer database is used. However, this file is here for completeness.

SEGMENT.DAT Relates segment (ID) to the site feeding it (inSite).

```
SegmentID,InSite
```

SegmentID	Segment name	30 characters
InSite	Name of site feeding segment	30 characters

The convention of the segment name being the in-site name prefixed by S- is not assumed in the software. However it is assumed in the a2s.csv file.

SEGMENT.OUT Relates segment (ID) to the sites it feeds (outSite).

```
SegmentID,OutSite
```

SegmentID	Segment name	30 characters
OutSite	Site fed by this segment	30 characters

SEGMENT.SUB Relates segment (ID) to the distribution transformers it feeds (subSite)

SegmentID, SubSite

SegmentID	Segment name	30 characters
SubSite	TXD site fed by this segment	30 characters

SITE.DAT Records relating to (earth) sites.

SiteID, Function, Location, Road, Area, Status

SiteID	(earth) Site name	10 characters
Function	Feeding, Network, Subsite or misc	characters
Location	Position of the site	characters
Road	Road site is on	characters
Area	Network area site is in	characters
Status	Open or Closed	1 character

Asset 2 Segment Link file:

This text file is situated in the .\A2S\ directory and is the database that relates the asset database to the Gentrack ENS database.

A2S.CSV

The segment name, an asset number on that segment, and the portion of the asset on that segment are the three columns of this file.

```
SegmentID,AssetID,Fraction
```

SegmentID	Segment name	30 characters
AssetID	Asset number	30 characters
Fraction	Fraction of this asset on this site	floating point number

Because this file came directly from the asset maps, the convention of the segment name being the in-site name prefixed by S- IS ASSUMED IN THIS FILE. This means that any non-standard entries in Gentrack (and there are some) will appear as errors in the Exceptions file.

Asset Database:

This is a comma-separated-value dump of the Alpine asset register.

ASSETS.CSV

It contains the dollar values of each asset number. Situated in the .\Alpine\ directory it contains the following fields:

```
AREA,CAT,ID_CODE,NUMBER,ASSET,LV%,REP_CODE,PH,
O/H,V,FED,CAB,A,C%,T,TX_CODE,X,P,DI,DM,TL,RL,
REP,REPU,RC,RDC,ORC,ODC,OPTIMISED,OPT_NOTE
```

Only a couple of these fields are used by the software. They are:

Area	Area the asset is in	10 characters
Cat	Category – HV,LV,SUB,SW	5 characters
ID_Code	Asset identification number	30 characters
TX_Code	TXD that the LV line feeds from	10 characters
RC	Cost to replace existing type of asset	long integer
ODC	Opt. Dep'cd RC w.r.t. age	long integer

(!) Even those items not used must be present to ensure the correct spacing of the remaining items in the list.

Two asset numbers are used internally by the program (via the a2s.csv file) and these must NOT BE ASSIGNED to any assets in the network, and therefore NOT INCLUDED IN THE ASSET DATABASE. They are:

- 00000 Which represents a segment that is known to have no assets attached to it. This prevents the segment being listed in the exception file.
- 99999 Which represents a segment in the TE area. This is used to group these segments so a standard (averaged) line charge may be applied to them due to the lack of detailed asset data.

Internal Databases:

Although the following two databases aren't externally created their field names are listed here for completeness. They are found in the .\progDATA\ directory.

LVTXD.DAT

Program created records relating to TXDs.

```
txdID,size,ODvalue,kVA,kWhr,lvODperKVA,lvODperKWhr,
hvODperKVA,hvODperKWhr,REPvalue,lvREPperKVA,
lvREPperKWhr,hvREPperKVA,hvREPperKWhr
```

TxdID	TXD site name	30 characters
Size	Rating of TXD	integer
ODvalue	$value of LV segment – ODV	unsigned long int
kVA	Installed kVA off this TXD	unsigned long int
kWhr	Actual kWhr off this TXD	unsigned long int
lvODperKVA	ODV value of LV assets / kVA	floating point num
lvODperKWhr	ODV value of LV assets / kWhr	floating point num
hvODperKVA*	ODV value of HV assets / kVA	floating point num
hvODperKWhr*	ODV value of HV assets / kWhr	floating point num
REPvalue	$value of LV segment –RC	unsigned long int
lvREPperKVA	RC value of LV assets / kVA	floating point num
lvREPperKWhr	RC value of LV assets / kWhr	floating point num
hvREPperKVA*	RC value of HV assets / kVA	floating point num
hvREPperKWhr*	RC value of HV assets / kWhr	floating point num

*HV values relate to the portion of the network utilised by this TXD as outlined in section 9.4.3.1.

SEGKVA.DAT

Program created records relating to segments.

```
SegmentID,size,ODvalue,kVA,kWhr,lvODperKVA,
lvODperkWhr,hvODperKVA,hvODperKWhr,REPvalue,
lvREPperKVA,lvREPperKWhr,hvREPperKVA,hvREPperKWhr
```

SegmentID	Segment name	30 characters
Size	zero	
ODvalue	$value of HV segment – ODV	unsigned long int
kVA*	Installed kVA off this segment	unsigned long int
kWhr*	Actual kWhr off this segment	unsigned long int
lvODperKVA	zero – no LV assets associated with HV segments	
lvODperKWhr	zero – no LV assets associated with HV segments	
hvODperKVA	ODV value of HV assets / kVA	floating point num
hvODperKWhr	ODV value of HV assets / kWhr	floating point num
REPvalue	$value of HV segment – RC	unsigned long int
lvREPperKVA	zero – no LV assets associated with HV segments	
lvREPperKWhr	zero – no LV assets associated with HV segments	
hvREPperKVA	RC value of HV assets / kVA	floating point num
hvREPperKWhr	RC value of HV assets / kWhr	floating point num

* As per section 9.4.3.1, this is the total load off this segment plus the load downstream of this segment.

Customer Databases:

AEL.CSV

The customer data file from United in comma separated value format. The following two files are found in the .\United\ directory.

```
Install,CSNO,Name,Address,Trans No,HV,kVA,Phases,
Tariffs,ANZIC,District,Error Code,
Summer kWh ,Winter kWh
```

Install	Unique customer number	long integer
CSNO*	Customer Service Number	30 characters
Name	Customer name	50 characters
Address	Address of customer	50 characters
Trans No	Name of TXD attached to	30 characters
HV	- not used(!)	
kVA	Installed capacity	long integer
Phases	- not used(!)	
Tariffs	- not used(!)	
ANZIC	- not used(!)	
District	- not used(!)	
Error Code	- not used(!)	
Summer kWhr	Energy units used last summer	long integer
Winter kWhr	Energy units used last winter	long integer

* This CSNO must be non-blank for an occupied installation as it is used to filter off vacant installs if it is blank

(!) Even those items not used must be present to ensure the correct spacing of the remaining items in the list.

CUSTOMER.DAT

The software processed United database with "pretty" aspects removed. The file structure remains the same.

DERIVATION OF LINE CHARGE ALGORITHMS

FOR

ALPINE ENERGY LIMITED

A.L.P. – ALPINE LINE-CHARGE PROGRAM.

SOFTWARE PROTOTYPES

By

CHRISTOPHER SAVAGE

UNIVERSITY OF CANTERBURY

DECEMBER, 1997

```
/************************************************
*  Alpine Line Charge Program                   *
*  define.h                                     *
*  data structures and type declerations        *
*  Written by Chris Savage                      *
*  v1.0 November 1997                           *
*  Last edited 12/11/1997                       *
************************************************/

#ifndef __includeFiles  // This #ifndef, #define, #endif sequence makes sure
#define __includeFiles  // that these #include statements are only performed once.

#include <windows.h>    // Standard Windows interface routines and declarations
#include <bwcc.h>       // Borland Windows Custom Controls used in screens of program
#include <stdio.h>      // File I/O procedures
#include <string.h>     // string comparison and manipulation procedures
#include <stdlib.h>     // Type conversion procedures
#include <alloc.h>      // Memory allocation and management procedures
#include <time.h>       // Date/Time stamp data structures and procedures

#include "calc.h"       // Line charge calculation algorythms
#include "check.h"      // Data integrity checking algorythms

   /* The following series of data (d_) manipulation files are divided into the following sections :
   d_high   : High level Search engines      => Using search engines to trace feeders/pos
   d_search : Search engines                 => for finding things in the csv files
   d_record : Char -> Record Transformations => for constructing records from csv file
   d_free   : Record Freeing routines        => to release malloc'd lists of data        */

#include "d_free.h"     // Data manipulation and search algorythms
#include "d_high.h"     // Data manipulation and search algorythms
#include "d_record.h"   // Data manipulation and search algorythms
#include "d_search.h"   // Data manipulation and search algorythms

#include "file.h"       // File I/O and searching algorythms
#include "find.h"       // Data find and report algorythms
#include "frontend.h"   // GUI operations and Windows interface
#include "network.h"    // Network structure definitions

#endif  // __includeFiles



#ifndef __dataStructures   // This #ifndef, #define, #endif sequence makes sure
#define __dataStructures   // that data structures are only defined once.

// The following sizes are used for data structure array length
#define wee    5
#define tiny  10  // leave this one! it's used for descrption & dollars in MasterLIST
#define small 30
#define med   35
#define large 50
#define MAXchar 200  // Maximum length of a line to be read in.


// Set file names for each part of the database
#define Seg2AssetFILE    ".\\a2s\\a2s.csv"         // relates segmentID:Asset#
#define EquipLnkFILE     ".\\ens\\equip.lnk"       // relates equipment:site
#define InstallLnkFILE   ".\\ens\\install.lnk"     // relates installation:TXD
#define SegSubFILE       ".\\ens\\segment.sub"     // relates segment:TXD
#define SegmentFILE      ".\\ens\\segment.dat"     // relates segmentID:inSite
#define SegOutFILE       ".\\ens\\segment.out"     // relates segmentID:outSite

#define segKvaFILE       ".\\progData\\segKVA.dat" // records key=segmentID
#define LVtxdFILE        ".\\progData\\LVtxd.dat"  // records key=TXD

#define InstallationFILE ".\\ens\\install.csv"     // records key=installation#
#define AssetDataFILE    ".\\alpine\\assets.csv"   // records key=asset#
#define EquipmentFILE    ".\\ens\\equip.dat"       // records key=equipmentID
#define SiteFILE         ".\\ens\\site.dat"        // records key=siteID
#define CustomerFILE     ".\\United\\customer.dat" // records key=Customer#
#define UnitedFILE       ".\\United\\AEL.csv"      // records key=Customer# - Unformatted

#define OutputDIR        ".\\output\\"             // Sets output directory

#define errorFILE        ".\\output\\error.txt"    // error messages are copied here.

#define iniFILE          ".\\alp.ini"              // Contains initalisation data.

#define helpFILE         ".\\help\\alpine.hlp"     // Online help file

#define tou1   "980"  // First 3 digits of a TOU customer number  //
#define tou2   "990"  // First 3 digits of a TOU customer number? // Used when removing TOU

/* The following declarations relate to the lists displayed in the drop-down boxes
   of the user interface by defining the names here as "int"s that match index
   into drop-down boxes allows easier reading of the main loops of this program.  */
```

```
// Resolution :
#define Global                  1
#define PointOfSupply           2
#define Feeder                  3
#define DistributionTransformer 4

// POS :
#define All         1
#define Albury      2
#define Balmoral    3
#define Clandeboye  4
#define Fairlie     5
#define Geraldine   6
#define Glentanner  7
#define GrasmereSt  8
#define Haldon      9
#define HuntSt     10
#define Pareora    11
#define PlPt       12
#define Studholme  13
#define Tekapo1052 14
#define Temuka     15
#define Timaru     16
#define Twizel6312 17
#define Unwin      18
#define VictoriaSt 19


// Type definitions used in this program

/* ==================== searchResultTYPE ==================== */

typedef struct searchResultTYPE
   // Allows construction of a linked list of lines that meet search criteria.

      {char line[MAXchar];
      struct searchResultTYPE* next;
      };

/* ======================== AssetDataTYPEs ========================== */

typedef struct AssetDataTYPE
   // Stores records form the asset database.

      {  char  idCode[small], // Asset identification number
               area[tiny],    // Area the asset is in
               cat[wee],      // Category - HV, LV, SUB, SW
//             asset[med],    // Asset code
//             repcode[tiny], // Replacement code
//             oh[wee],       // Overhead or Underground
//             fed[wee],      // Feeder asset origionates from
               txCode[tiny];  // Tx that LV line feeds from
//             optimised,     // Is asset value optimised?
//             optNote[large],// If asset is value optimised, comments
//             A,             // 'A' if u/g cable has aluminium core
//             T,             // Terrain (Flat, Rolling, Mountainous, Urban)
//             P;             // 'P' if pilot wire present
//       float number;        // Length of line or number of assets
//       int   LVpercent,     // % of route length that LV shares HV pole
//             phase,         // Number of phases 1/2/3 being recorded
//             volts,         // Operational voltage 33kV, 11kV, 400V, 230V
//             cab,           // U/G cable core size
//             Cpercent,      // Percentage of concrete poles in asset length
//             X,             // Number of LV Road crossings in asset span
//             DI,            // Year that asset was installed
//             DM,            // Year o/h line was last maintained
//             TL,            // Normal expected life of asset
//             RL;            // Remaining life of asset
//    long int REP;           // Replacement cost per unit (present)
//             REPU,          // Ditto, but sharing with another cable pole/trench
      long int RC;           // Calc'd cost to replace existing type of asset (PV)
//             RDC,           // Replacement depreciated cost w.r.t age
//             ORC,           // The cost to replace the asset with optimum choice (today)
      long int ODC;          // Opt. Depreciated replacement cost w.r.t age
      float    fractionUsed; // By Chris for storing % used on this segment
      };


typedef struct AssetDataListTYPE
   // Allows construction of a linked list of AssetData records

         {  struct AssetDataListTYPE *next;
            struct AssetDataTYPE     *record;
         };

/* ======================== Seg2AssetTYPEs ======================== */

typedef struct Seg2AssetTYPE
```

```
   // Stores records from the segment name : asset idCode file.

         {  char  segment[small],   // Segment identification code
                  idCode[small];    // Asset code for segment (1:n)

            float fractionUsed;     // Percentage of asset inside this segment
         };

typedef struct Seg2AssetListTYPE
   // Allows construction of a linked list of Seg2Asset records

         {  struct Seg2AssetListTYPE *next;
            struct Seg2AssetTYPE     *record;
         };

/* ============================ SiteTYPE ============================== */

typedef struct SiteTYPE
   // Stores a gentrack site record

         {  char  siteID[tiny],  // Site identification code
                  function[],    // Site function
                  location[],    // Location of site
                  road[],        // Road site is on/near
                  area[],        // Area that site is in
                  status;        // Open/Closed status of site
         };

/* ======================== InstallationTYPEs ======================== */

typedef struct InstallationTYPE
   // Stores a United Electricity installation record

         {  long  accountNo,     // Customer account number
                  kVArating,     // Installed capacity

                  summerKWhr,    // Summer usage
                  winterKWhr,    // Winter Usage
                  kWhr;          // = Summer + Winter

            float AssetDollars;  // Dollar portion of the network utilised

            float fixedYr,       // Fixed yearly line charge to be paid
                  varYr,         // Variable line charge to be paid
                  perKWhr,       // The rate at which the variable rate is charged
                  oldLC,         // Line charge under current system
                  percentInc;    // Change in LC from old sysrem to new.

            char  siteID[small]; // Identification code of transformer fed from
         };

typedef struct InstallationListTYPE
   // Allows construction of a linked list of Inst2TXD records

         {  struct InstallationListTYPE *next;
            struct InstallationTYPE     *record;
         };


/* ========================= Equipment ============================ */

typedef struct EquipmentTYPE
   // Stores a gentrack equipment record

         {  char  IDcode[small],  // Equipment identification code
                  Class[wee],     // Equipment class
                  type[wee],      // Type of equipment
                  vRatio[small],  // Voltage ratio (of Tx)
                  serialNo[small];// Serial number of equipment if appropriate

            int   kVArating,      // kVA rating of equipment
                  phases;         // Number of phases

            float purchaseCost;   // If known - not used in calculations
         };

typedef struct EquipmentListTYPE
   //

         {  struct EquipmentListTYPE *next;
            struct EquipmentTYPE     *record;
         };

/* ========================= Segments ============================ */

typedef struct Seg2SiteTYPE
   // Stores segmentID : InSite  information
   //        segmentID : OutSite information
```

```
   //        segmentID : Subsite information

         {  char  segmentID[small], // Segment identification code
                  siteID[small];    // Site identification code
         };


typedef struct Seg2SiteListTYPE
   // Allows construction of a linked list of Segment2Site records

         {  struct Seg2SiteListTYPE *next;
            struct Seg2SiteTYPE     *record;
         };

/* ========================= CalcDataTYPE ========================== */

typedef struct CalcDataTYPE
   // Stores data on segments from extended data file

         {  char  ID[small];  // Segment (or TXD) identification code

            int   size;       // Rating of TXD

   unsigned long  ODvalue,    // $value of HV/LV segment - depreciated
                  REPvalue,   // $value of HV/LV segment - replacement
                  kVA,        // Installed kVA off this segment/TXD
                  kWhr;       // Actual kWhr off this segment/TXD

            float lvODperKVA,    // ODV value of LV asset (per KVA)
                  lvODperKWhr,   // ODV value of LV asset (per kWhr)

                  hvODperKVA,    // ODV value of HV asset (per KVA)
                  hvODperKWhr;   // ODV value of HV asset (per kWhr)

            float lvREPperKVA,   // REP value of LV asset (per KVA)
                  lvREPperKWhr,  // REP value of LV asset (per kWhr)

                  hvREPperKVA,   // REP value of HV asset (per KVA)
                  hvREPperKWhr;  // REP value of HV asset (per kWhr)
         };


typedef struct CalcDataListTYPE
   // Allows construction of a linked list of Segment2Site records

         {  struct CalcDataListTYPE *next;
            struct CalcDataTYPE     *record;
         };

/* ========================= MasterListTYPE ========================== */

typedef struct MasterListTYPE
   // A pointer to the heads of all the lists read in from files.

         {  struct Seg2AssetListTYPE *Seg2AssetL;
               // Seg2AssetFILE -> relates segmentID:Asset#

            struct InstallationListTYPE *InstallationL;
               // CustomerFILE -> records : key=customer#

            struct Seg2SiteListTYPE  *SubL,     // SegSubFILE -> relates segmentID:TXD
                                     *OutL,     // SegOutFILE -> relates segmentID:outSite
                                     *SegL,     // SegmentFILE -> relates segmentID:inSite
                                     *eLnkL;    // EquipLnkFILE -> relates site:equipment

            struct AssetDataListTYPE *AssetL;   // AssetDataFILE -> records : key = asset#

            struct CalcDataListTYPE  *segKVA,   // segKvaFILE -> records : key = segmentID
                                     *LVtxd;    // LVtxdFILE -> records :  key = TXD

            struct EquipmentListTYPE  *EquipL;
               // EquipmentFILE & EquipLnkFILE -> records : key=equipmentID


            char FileNAME[large];   // Name of output file - calculated from inputs

            // Stores data about breakdown of dollar requirements and how to divide it up
            char  names[tiny][small];  // Description of fields
            int   dollars[tiny];       // amounts for each field
            int   fixed[tiny];         // Fixed or Variable allocation

            int   fixedDollars;        // Total $ to be split on fixed basis
            int   variableDollars;     // Total $ to be split on variable basis

            // Old line charges information.

            float oldLC[3][4];         // 3x4 table containing kVA, fixed, perKW, perKWhr
                                       // for <15KVA, 45.3kVA, all else.
```

```
        int   AllocatedODVassets;  // Total amount of assets allocated in the system
        int   AllocatedREPassets;  // Total amount of assets allocated in the system
        int   numConsumers;        // Total number of consumers connected to system

        // The Tu. City is treated differently.  The TOTAL value of assets is used,
        // rather than per segment

        int TimaruODVassets;  //
        int TimaruREPassets;  // Loaded from the INI file

        BOOL  odv;  // Data to base calculations on: ODV (true) or REP (false) cost
        BOOL  tou;  // Whether or not 980 customers are included in the customer database
    };

/* ========================= END define.h ========================== */

#endif  // __dataStructures
```

```
/************************************************
*  Alpine Line Charge Program                   *
*  calc.h                                       *
*  Interfaces between the GUI and med/low level *
*     search engines of the program             *
*  Written by Chris Savage                      *
*  v1.1 December 1997                           *
*  Last edited 13/11/1997                       *
************************************************/

void calculate(char *POS, char *FeederID, char *Resolution, char *U15, char *P15, HWND hMain,
               struct MasterListTYPE *MasterLIST);
   // Interface between the GUI and the medium-level procedures of the program.

void calcAll(char *Resolution, char *U15, char *P15, HWND hMsg, struct MasterListTYPE *MasterLIST);
   // Special case of All, All selected

void calcTXDfeeder(  char *FeederID, char *U15, char *P15, HWND hMsg,
                     struct MasterListTYPE *MasterLIST);
   /* Performs line charge calculation, allocating individual line charges to each TXD
      on the feeder (or POS). */

void setupFeederCalc(char *POS, char *FeederID, char *U15, char *P15, HWND hMsg,
                     struct MasterListTYPE *MasterLIST);
   /* Perform calcAVGEfeeder procedure calls to get individual sets of line charges
      for each feeder when the whole substation is selected. */

void calcAVGEfeeder( char *FeederID, char *U15, char *P15, HWND hMsg,
                     struct MasterListTYPE *MasterLIST);
   /* Performs line charge calculation, allocating THE SAME line charges to each TXD
      on the feeder (or POS). */

void calcGLOBALfeeder(char *U15, char *P15, HWND hMsg, struct MasterListTYPE *MasterLIST);
   /* Assigning the same line charge (rate per KVA) to each TXD in the system */

struct MasterListTYPE* rebuildDatabase(HWND hMsg, struct MasterListTYPE *MasterLIST);
   // Uses ENS and Asset-2-segment data to create a new file with segment statistics in it

struct MasterListTYPE* rebuildLVtxd(HWND hMsg, struct MasterListTYPE *MasterLIST);
   /* Calculates the dollar value and loading on LV segments   */


struct MasterListTYPE* rebuildSegKVA(HWND hMsg, struct MasterListTYPE *MasterLIST);
   /* Uses the data from United and Gentrack to a new data file.

      This is a file that calculates how many (kWhr, kVA) are dependant on
      each (HV) segment. This is the total of the load of this segment, and all segments
      that feed from it.
      The dollar value of the segment is also calculated.   */

void CalcDollarsPer(HWND hMsg, struct MasterListTYPE *MasterLIST);
   /* Calculates the hv assets used (per KVA and kWhr) by each TXD and stores
      them in the appropriate place in the data structure*/

struct InstallationListTYPE *CalcDollars( struct Seg2SiteListTYPE *segments, HWND hMsg,
                                          struct MasterListTYPE *MasterLIST);
   /* Given a list of segments, calculate the dollars of HV and LV assets used
      by each installation off these segments. */

void CalcTimaru(HWND hMsg, struct MasterListTYPE *MasterLIST);
   /* Calculates the assets used by each of the consumers in the Timaru City area.
      This is simply the total assets in the Tu City / number of consumers in Tu City */

void CalcOldLC(HWND hMsg, struct InstallationListTYPE *InstallationList,
               struct MasterListTYPE *MasterLIST);
   /* Given a list of installations, calculate the line charge under the existing
      system, and the % variance of using the new method.  */
```

```
/************************************************
*  Alpine Line Charge Program                   *
*  check.h                                      *
*  Data integraty checking operations - headder *
*  Written by Chris Savage                      *
*  v1.0 November 1997                           *
*  Last edited 11/11/1997                       *
************************************************/

void Debug(HWND hwndMain, HANDLE hInst, HWND hMsg, struct MasterListTYPE *MasterLIST);
   /* Called by Test:Debug menu to test new functions */

void CheckSeg2A(HWND hMsg, struct MasterListTYPE *MasterLIST);
   /* Checks the segment-2-asset data file against the site-2-segment
      file, and reports exceptions.
      (segments that are listed in one, but not the other) */

void CheckAssetAllocation(HWND hMsg, struct MasterListTYPE *MasterLIST);
   /* Checks the segment-2-asset data file against the Asset data
      file, and reports exceptions.
      (Assets that are listed in one, but not the other) */

void CheckS2Anumbers(HWND hMsg, struct MasterListTYPE *MasterLIST);
   /* Checks the fractions in the segment-2-asset data file to make sure
      each asset number has ~1.0 usage, and reports exceptions.  */
```

```c
/*****************************************************
*  Alpine Line Charge Program                        *
*  d_free.h                                          *
*  Data type operations                              *
*  Written by Chris Savage                           *
*  v1.0 November 1997                                *
*  Last edited 13/11/1997                            *
*****************************************************/

/* This series of data (d_) manipulation files are divided into the following sections :

   d_high :   High level Search engines      => using search engines to trace feeders/pos
   d_search : Search engines                 => for finding things in the csv files
   d_record : Char -> Record Transformations => for constructing records from csv file
   d_free :   Record Freeing routines        => to release malloc'd lists of data

*/

// §§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§ Record freeing routines

void freeSearchResult(struct searchResultTYPE *searchResultPTR);

   /* Frees space occupied by a searchResult list */

void freeInstallationList(struct InstallationListTYPE *InstallationListPTR);

   /* Frees the space used by an Inst2TXDList */

void freeSeg2SiteList(struct Seg2SiteListTYPE *Seg2SiteListPTR);

   /* Frees space used by a Seg2SiteList */

void freeAssetDataList(struct AssetDataListTYPE *AssetDataListPTR);

   /* Frees space occupied by an AssetDataList */

void freeSegment2AssetList(struct Seg2AssetListTYPE *Segment2AssetListPTR);

   /* Frees space occupied by a Segment2AssetList */

void freeCalcDataList(struct CalcDataListTYPE *CalcDataListPTR);

   /* Frees space occupied by a CalcDataList */

void freeMasterList(struct MasterListTYPE *ML);

   /* Frees the space used by an MasterList
      July > DOES!! delete data loaded as list at start */
```

```
/****************************************************
*  Alpine Line Charge Program                      *
*  d_high.h                                        *
*  Data type operations                            *
*  Written by Chris Savage                         *
*  v1.0 November 1997                              *
*  Last edited 13/11/1997                          *
****************************************************/

/* This series of data (d_) manipulation files are divided into the following sections :

   d_high :   High level Search engines      => using search engines to trace feeders/pos
   d_search : Search engines                 => for finding things in the csv files
   d_record : Char -> Record Transformations => for constructing records from csv file
   d_free :   Record Freeing routines        => to release malloc'd lists of data

*/


// §§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§ High Level Search engines

struct Seg2SiteListTYPE* traceSource(char *site, HWND hMsg, struct MasterListTYPE *MasterLIST);

   /* Given the ID of a (TXD) site, return a linked list of all the segments that
      feed to it, back to the POS in the form of Seg2Site records.   */


struct Seg2SiteListTYPE* getDependants(char *FeederID, HWND hMsg,
                                       struct MasterListTYPE *MasterLIST);

   /* Given the ID of a site, return a linked list of all the segments fed
      from it. */

struct Seg2SiteListTYPE* getTXDs(struct Seg2SiteListTYPE* seg2siteList, HWND hMsg,
                                 struct MasterListTYPE *MasterLIST);

   /* Given the linked list of siteIDs, return a linked list of all the TXDs fed
      from them in the form of Seg2Site records. */

struct InstallationListTYPE* getInstallations(  struct Seg2SiteListTYPE* txdList, HWND hMsg,
                                                struct MasterListTYPE *MasterLIST);

   /* Given the linked list of TXDs, return a linked list of all the Installations fed
      from them in the form of Inst2TXD records. */

struct Seg2SiteListTYPE* getAllSegments(HWND hMsg, struct MasterListTYPE *MasterLIST);

   /* Returns a linked list of all segments currently linked into the Gentrack
      records. */

int getTXDsize(char *site, struct MasterListTYPE *MasterLIST);

   /* Given the name of a distribution transformer site, return the size (kVA)
      of (the total of all) distribution transformers on the site */

void getDollars(  struct Seg2SiteListTYPE* segList, struct CalcDataTYPE *txdRecord, HWND hMsg,
                  struct MasterListTYPE *MasterLIST);

   /* Given that CalcData record relating to a txdRecord, and the list
      of segments that connect it back to a POS, fill in the totalKWhr and totalKVA
      fields of the txdRecord. */

int getNumberOfConsumers(HWND hMsg, struct MasterListTYPE *MasterLIST);

   /* Return an integer value of the number of consumers currently
      connected into the network. */

int getAllocatedODVassets(HWND hMsg, struct MasterListTYPE *MasterLIST);

   // returns the dollar value of allocated assets in the system

int getAllocatedREPassets(HWND hMsg, struct MasterListTYPE *MasterLIST);

   // returns the dollar value of allocated assets in the system

int getFeederODVassets(HWND hMsg, struct MasterListTYPE *MasterLIST, char *feeder);

   // returns the dollar value of allocated assets on the given feeder

int getFeederREPassets(HWND hMsg, struct MasterListTYPE *MasterLIST, char *feeder);

   // returns the dollar value of allocated assets on the given feeder

int getPOSnumber(char *POS);

   // Given the string containing the name of a POS, return it's number in the array

int getRESnumber(char *Resolution);
```

```
   // Given the string containing the name of a Resolution, return it's number in the array

float getMETHODpc(char *Method);

   // Given the string containing the name of a Method, return it's matching %

void getConsumerAndAssets(HWND hMsg, struct MasterListTYPE *MasterLIST);

   /* Fill in the spaces in the MasterLIST for number of consumers an dollar value
      of assets in the network */
```

```c
/***************************************************
*  Alpine Line Charge Program                      *
*  d_record.h                                      *
*  Data type operations                            *
*  Written by Chris Savage                         *
*  v1.0 November 1997                              *
*  Last edited 13/11/1997                          *
***************************************************/

/* This series of data (d_) manipulation files are divided into the following sections :

   d_high :   High level Search engines      => using search engines to trace feeders/pos
   d_search : Search engines                 => for finding things in the csv files
   d_record : Char -> Record Transformations => for constructing records from csv file
   d_free :   Record Freeing routines        => to release malloc'd lists of data

*/


// §§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§ Char -> Record Transformations


struct SiteTYPE* SiteRecord(char *data);

   /* Given an array of char containing csv's relating to a Site record,
      create the record in struct format and return a poiner to it.  */

struct InstallationTYPE* InstallationRecord(char *data);

   /* Given an array of char containing csv's relating to an Inst2TXD
      record, creates the record in struct format and returns a pointer to it.*/

struct EquipmentTYPE* EquipmentRecord(char *data);

   /* Given an array of char containing csv's relating to an equipment
      record, creates the record in struct format and returns a pointer to it.*/

struct Seg2SiteTYPE* Seg2SiteRecord(char *data);

   /* Given an array of char containing csv's relating to a Seg2Site
      (In, Out or SUB site) record, creates the record in struct format and
      returns a pointer to it.   */

struct Seg2AssetTYPE* Segment2AssetRecord(char *data);

   /* Given an array of char containing csv's relating to a Segment2Asset
      record, creates the record in struct format and returns a pointer to it.*/

struct AssetDataTYPE* AssetDataRecord(char *data);

   /* Given an array of char containing csv's relating to an asset record,
      creates the record in struct format and returns a pointer to it.*/

struct CalcDataTYPE* CalcDataRecord(char *data);

   /* Given an array of char containing csv's relating to a CalcData
      record, creates the record in struct format and returns a pointer to it. */
```

```c
/****************************************************
*  Alpine Line Charge Program                       *
*  d_search.h                                       *
*  Data type operations                             *
*  Written by Chris Savage                          *
*  v1.0 November 1997                               *
*  Last edited 13/11/1997                           *
****************************************************/

/* This series of data (d_) manipulation files are divided into the following sections :

   d_high :   High level Search engines      => using search engines to trace feeders/pos
   d_search : Search engines                 => for finding things in the csv files
   d_record : Char -> Record Transformations => for constructing records from csv file
   d_free :   Record Freeing routines        => to release malloc'd lists of data

*/


// §§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§ Search engines


struct InstallationTYPE* Installation2TXD(char *InstallationID, struct MasterListTYPE *MasterLIST);

   /* Given the Installation #, return the INST2TXD record that relats it to
      a distribution transformer.   */

struct InstallationListTYPE* TXD2Installation(  char *TXDnumber, HWND hMsg,
                                                struct MasterListTYPE *MasterLIST);

   /* Given a distribution transformer number, return a pointer to the head of a
      linked list containing Installation:TXD's relating to it.   */

struct Seg2SiteListTYPE* Seg2TXD(char *segmentID, struct MasterListTYPE *MasterLIST);

   /* Given the name of a segment, return a pointer to the top of a linked list of
      Seg2Site records giving the SubsSites off this segment (subSites = TXD's). */

struct Seg2SiteListTYPE* Site2Equip(char *siteID, struct MasterListTYPE *MasterLIST);

   /* Given the name of a site, return a pointer to the top of a linked list of
      Seg2Site records giving the equipment off this site.  */

struct Seg2SiteTYPE* TXD2Seg(char *TXDsite, struct MasterListTYPE *MasterLIST);

   /* Given the name of a TXD, return the Seg2Site record that relates it to a segment.*/

struct Seg2SiteTYPE* Seg2In(char *segmentID, struct MasterListTYPE *MasterLIST);

   /* Given the name of a segment - give the Seg2Site record that relates it to
      the site feeding it. */

struct Seg2SiteTYPE* Site2In(char *site, struct MasterListTYPE *MasterLIST);

   /* Given the name of a site - give the Seg2Site record that relates it to
      the segment feeding it. */

struct Seg2SiteTYPE* Site2Seg(char *siteID, struct MasterListTYPE *MasterLIST);

   /* Given the name of a site, give the Site2Seg record that gives the
      name of the segment that it feeds   */

struct Seg2SiteListTYPE* Seg2Out(char *segmentID, struct MasterListTYPE *MasterLIST);

   /* Given the name of a segment, return a pointer to the head of a linked
      list of site2seg records that ot feeds.   */

struct AssetDataListTYPE* getAssets(char *segment, struct MasterListTYPE *MasterLIST);

   /* Given the name of a segment, return a linked list of the (HV line) asset
      records (assetData records) that are included in it.  */

struct Seg2AssetListTYPE* getAssetConnection(char *asset, struct MasterListTYPE *MasterLIST);

   /* Given the name of an asset, return a linked list of the segment
      numbers that it is distibuted across.  */

struct AssetDataListTYPE* getLVassets(char *TXD, struct MasterListTYPE *MasterLIST);

   /* Given the name of a TXD, return a linked list of the (LV line) asset
      numbers that are included in it. */

struct CalcDataTYPE* getTxdRecord(char *siteID, struct MasterListTYPE *MasterLIST);

   /* Given the name of a TXD site, give the CalcData record that gives the
      record relating to this site*/

struct CalcDataTYPE* getSegRecord(char *segID, struct MasterListTYPE *MasterLIST);
```

```
   /* Given the name of a segment, give the CalcData record that contains the
      record relating to this site*/

char *getCSVitem(char *word, int itemNum);

   /* Given an array of char containing a line of comma separated values,
      returns the 'itemNum' value in the list.  */
```

```
/************************************************
*  Alpine Line Charge Program                   *
*  file.h                                       *
*  File I/O and search operations               *
*  Written by Chris Savage                      *
*  v1.0 November 1997                           *
*  Last edited 17/11/1997                       *
*     Added createTekapoLink, LoadNewInstalls   *
************************************************/


void makeFileName(char *POS,           // Point of Supply into network chosen for this run
                  char *FeederID,      // Feeder identification name/number
                  char *Resolution,    // How detailed the calculations are
                  char *U15,           // The method used to assign costs - based on usage
                  char *P15,
                  struct MasterListTYPE *MasterLIST);

   /* Create unique-ish output file name given the input parameters. The structure of the
       file name is :

           The first (3) letters of the POS name
         + The last (2) letters of the Feeder name
         + The first (1) letter of the Resolution name
         + The first (1) letter of the method
         + .txt on the end.
   */


struct searchResultTYPE* search( char *inputFileNAME, // Name (and path) of the file to
                                                      //  find information in
                                 char *searchSTRING); // String to search for in file

   /* This program (linear) searches the given input file for a line containing
      a given search string as a substring. Matching lines are returned as a linked list
      of SearchResultTYPE.
   */

void WriteToFile( char   *fileNAME,
                  struct Seg2AssetListTYPE      *S2A,
                  struct InstallationListTYPE   *I2T,
                  struct Seg2SiteListTYPE       *S2S,
                  struct AssetDataListTYPE      *AD,
                  struct CalcDataListTYPE       *CDL,
                  struct EquipmentListTYPE      *EL);

   /* Write (append) any given list types to the output file provided.
      Does not alter lists or pointers.  Must be passed NULL for un-wanted lists. */

struct MasterListTYPE* ReadInFiles(HWND hMsg);

   /* Reads in all of the data into a MasterListTYPE and returns a pointer to the
       Mster List. This is called at the start of the program, and from this point all
       access to data is from RAM. (speeds things up)
   */

struct MasterListTYPE* formatUnited(HWND hMsg, struct MasterListTYPE *MasterLIST);

   /* Removes any "" marks that seem to appear around some numerical values in the
      United customer database (csv) file.  Called from the Database : Format United menu
   */

void WriteResult( char   *fileNAME,
                  struct InstallationListTYPE  *IL,
                  char   *FeederID,
                  int    feederAssets,
                  int    allocatedAssets,
                  float  feederFraction,
                  float  dollars2get,
                  float  factor,
                  float  fixedPortion,
                  char   *U15,
                  char   *P15,
                  char   *Resolution,
                  BOOL   odv);

   /* Write (append) the results of a calculation run to the output file (name)provided.
      Does not alter lists or pointers */

void createTekapoLink(void);
   // Links the Unwin, Glentanner and Haldon subs into Tekapo in the Genterack records

void LoadNewInstalls(void);
   // Links any non-United customers into the database
```

```
/************************************************
*  Alpine Line Charge Program                   *
*  find.h                                       *
*  Data searching operations                    *
*  Written by Chris Savage                      *
*  v1.0 November 1997                           *
*  Last edited 12/11/1997                       *
************************************************/

void WriteText(HWND hDlg, int dlgBOX, char *string);
   /* Given a line of text, appends it to the end of the existing text in the specified
      text box */

void InstallationInfo(struct MasterListTYPE *MasterLIST, HWND hDlg, char *string);
   /* Given the number of an Installation, provides information on the installation itself
      and the way it connects into the network. */

void TxdInfo(struct MasterListTYPE *MasterLIST, HWND hDlg, char *string);
   /* Given the name of a Distribution Transformer, provides information on the
      TXD itself and the way it connects into the network. */

void SiteInfo(struct MasterListTYPE *MasterLIST, HWND hDlg, char *string);
   /* Given the name of an earth site, provides information on the site itself
      and the way it connects into the network. */

void SegmentInfo(struct MasterListTYPE *MasterLIST, HWND hDlg, char *string);
   /* Given the name of a segment, provides information on the segment itself
      and the way it connects into the network. */

void AssetInfo(struct MasterListTYPE *MasterLIST, HWND hDlg, char *string);
 /* Given the name of an asset, provides information on the asset itself
      and the way it connects into the network. */
```

```
/*************************************************
*  Alpine Line Charge Program                    *
*  frontend.h                                    *
*  GUI Operations and Windows interface          *
*  Written by Chris Savage                       *
*  v1.0 November 1997                            *
*  Last edited 12/11/1997                        *
*************************************************/

// The following #define statements simply assign interger values to each of the
//  buttons, lists, boxes and windows used in this program.

#define ICON_1             1  // The A.L.P Icon associated with the program
#define MENUBAR            1  // Menu bar on A.L.P. main screen

#define IDCANCEL           2

#define SOLVE_P15         85
#define SOLVE_POS         86
#define SOLVE_FeederID    87  // Text boxes associated with the "continue?" screen
#define SOLVE_Resolution  88
#define SOLVE_U15         89
#define SOLVE_DB          90
#define SOLVE_TOU         91

#define DB_FeederID       10
#define DB_U15            11
#define DB_Resolution     12
#define DB_POS            13  // Lists and buttons in the screen for entering settings
#define DB_P15            14
#define DB_ODV            15
#define DB_RC             16

#define MAIN_SETTINGS     20
#define MAIN_FIND         21
#define EXIT              22  // The buttons on the main screen of the program
#define MAIN_MSG          23
#define MAIN_DMY          24

#define FILENAME_NAME     25  // The text box in screen to check file name

#define MENU_RBLD         30
#define MENU_CHK          31
#define MENU_FORMAT       32  // Options available from the menu bar
#define MENU_TEKAPO       33
#define MENU_TEST         34
#define MENU_HELP         35
#define MENU_INSTALLS     36

#define FIND_ITEM         40
#define FIND_CONNECTION   41
#define FIND_ASSET        42
#define FIND_SEGMENT      43
#define FIND_SITE         44
#define FIND_TXD          45  // Text boxes associated with the "find" screen
#define FIND_INSTALLATION 46
#define FIND_STRING       47
#define FIND_GO           48


#define DOL_DESCRIPTION0  50
#define DOL_DESCRIPTION1  51
#define DOL_DESCRIPTION2  52
#define DOL_DESCRIPTION3  53
#define DOL_DESCRIPTION4  54
#define DOL_DESCRIPTION5  55  // Description field boxes in dollar allocation screen
#define DOL_DESCRIPTION6  56
#define DOL_DESCRIPTION7  57
#define DOL_DESCRIPTION8  58
#define DOL_DESCRIPTION9  59

#define DOL_VALUE0        60
#define DOL_VALUE1        61
#define DOL_VALUE2        62
#define DOL_VALUE3        63
#define DOL_VALUE4        64
#define DOL_VALUE5        65  // Dollar field boxes in dollar allocation screen
#define DOL_VALUE6        66
#define DOL_VALUE7        67
#define DOL_VALUE8        68
#define DOL_VALUE9        69


#define DOL_CV0           70
#define DOL_CV1           71
#define DOL_CV2           72
#define DOL_CV3           73
#define DOL_CV4           74
```

```
#define DOL_CV5           75
#define DOL_CV6           76  // Corporate/Variable flag boxes in dollar allocation screen
#define DOL_CV7           77
#define DOL_CV8           78
#define DOL_CV9           79
#define DOL_Total         80

#define DOL_SAVE          81  // The "save dollars" button on the dollar allocation screen


long FAR PASCAL _export MainWndProc (HWND, UINT, UINT, LONG) ;
    // The program window - Sets up and controls the main window of the program

BOOL FAR PASCAL _export SettingsDlgProc(HWND hDlg, UINT message, UINT wParam,LONG lParam);
    // The main program dialogue box window. This is where the network settings for the
    //   program are set.

BOOL FAR PASCAL _export ContinueDlgProc(HWND hDlg, UINT message, UINT wParam,LONG lParam);
   // The pop-up box that confirms the settings (chosen in the Settings box)

BOOL FAR PASCAL _export FindDlgProc(HWND hDlg, UINT message, UINT wParam, LONG lParam);
   // This procedure controlls the Find dialogue box.

BOOL FAR PASCAL _export DollarDlgProc(HWND hDlg, UINT message, UINT wParam, LONG lParam);
   // This procedure controlls the dollar dialogue box.

BOOL FAR PASCAL _export FileNameDlgProc(HWND hDlg, UINT message, UINT wParam, LONG lParam);
   // This procedure controlls the File Name dialogue box.

void SetUpWindow(HWND hDlg);
   // Function calls performed when the "settings" screen is first opened

void message(char type[], char location[], char message[]);
   /* Allows other parts of the program to send messages to the screen that are recorded
      them in the error.txt file. */

void updateStatus(void);
   // Shows the status bar along the bottom of the main window with the chosen parameters.

void LoadINI(struct MasterListTYPE *MasterLIST);
   // Loads options from the alp.ini file at the start of the program

void SaveDollars(struct MasterListTYPE *MasterLIST);
   // Saves dollar initilisation data to the .ini file associated with the program.
```

```
/*************************************************
*  Alpine Line Charge Program                    *
*  network.h                                     *
*  Include file listing the basic network POS    *
*  structure                                     *
*  Written by Chris Savage                       *
*  v1.0 November 1997                            *
*  Last edited 11/11/1997                        *
*************************************************/

// Array containing names of all POS.  Element (0) contains number of POS listed

char* pos[]={"19", "All", "Albury", "Balmoral", "Clandeboye", "Fairlie",   "Geraldine",
             "Glentanner",
             "Grasmere",  "Haldon", "Hunt",     "Pareora",    "PlPt",      "Studholme",
             "Tekapo1052","Temuka", "Timaru",   "Twizel6312", "Unwin",     "Victoria"};

// The same format is used for each of the following arrays of feeder names for various POS

char* alburyFEEDERS[] = {"5", "All", "Albury1", "Albury2", "Albury3", "Albury4"};

char* clandeboyeFEEDERS[] = {"11", "All",  "T601", "T602", "T603", "T604", "T605", "T609",
                             "T610", "T611", "T612", "T613"};

char* grasmereStFEEDERS[] = {"15", "All",  "TEGrasmere2",  "TEGrasmere4",  "TEGrasmere5",
                             "TEGrasmere6",  "TEGrasmere7",  "TEGrasmere9",  "TEGrasmere10",
                             "TEGrasmere12", "TEGrasmere13", "TEGrasmere15", "TEGrasmere16",
                             "TEGrasmere17", "TEGrasmere18", "TEGrasmere20"};

char* huntStFEEDERS[] = {"11", "All", "TEHunt1",  "TEHunt2",  "TEHunt4",  "TEHunt5",
                         "TEHunt7",   "TEHunt10", "TEHunt11", "TEHunt13", "TEHunt14",
                         "TEHunt16"};

char* studholmeFEEDERS[] = {"6", "All", "Studholme1", "Studholme2", "Studholme3",
                                        "Studholme4", "Studholme5"};

char* tekapoFEEDERS[] = {"7", "All", "M200", "M201", "M205", "M206", "M207", "M11"};

char* temukaFEEDERS[] = {"8", "All", "Temuka1", "Temuka2", "Temuka3", "Temuka4", "Temuka5",
                         "Temuka6",  "Temuka7"};

char* timaruFEEDERS[] = {"14", "All", "Timaru1",  "Timaru2",  "Timaru3",  "Timaru4",
                                      "Timaru5",  "Timaru6",  "Timaru11", "Timaru12",
                                      "Timaru13", "Timaru14", "Timaru15", "Timaru16",
                                      "Timaru17"};

char* twizelFEEDERS[] = {"6", "All", "Z1", "Z2", "Z3", "Z4", "Z9"};

char* victoriaStFEEDERS[] = {"9", "All",     "TEVictoria1", "TEVictoria3", "TEVictoria5",
                             "TEVictoria6",  "TEVictoria8", "TEVictoria9", "TEVictoria11",
                             "TEVictoria13"};

// The same format is used for the following parameters that may be selected within the program

char* RESOLUTION[] = {"4","Global", "Point of Supply", "Feeder", "Distribution Transformer"};

char* METHOD[] = {"11", "Fixed", "90", "80", "70", "60", "50", "40", "30", "20", "10",
                        "Variable"};
```

Appendix D: CHANGE OF POS PROCEDURE

The various Points of Supply (POS) around the network are assumed to be fixed. For this reason they are wired directly into the source code of the software. This allows the user interface to accept the various input parameters without having to check the validity of the input data.

One problem that arrises from this decision is that changing the POS structure of the network will require that the software code is adjusted and re-compiled.

During the course of this project the Clandeboye upgrade resulted in a new POS being created at Clandeboye. The software was adjusted to accept this change and the steps taken were documented.

The steps for adding a POS with more than one feeder is outlined below. However, changes required for a POS with only one feeder are noted. Removing a POS will simply require the opposite actions at each step of the process.

Adding a multi-feeder POS to the network:

1. All names and terms relating to the new POS are those that are used in Gentrack ENS.

2. Once the POS has been changed in Gentrack ENS, a new set of data files that reflect the new network will need to be extracted. The procedure for this is outlined in Appendix B:

3. After the existing data files are backed up, the new data files will have to be loaded into their correct directories. *Note that a new copy of the customer or asset database is not required for this action. Changing the physical structure of the network will not impact on the customer file – they will still be attached to the same segments (as long as site & segment names are not changed.)

4. In the software source code file network.h – add the new POS to the list titled "char* pos[] ="

```
char* pos[]={"19", "All", "Albury", "Balmoral", "Clandeboye", "Fairlie",
"Geraldine", "Glentanner", etc...
```

FIGURE 21 CODE FRAGMENT – POS DATA STRUCTURE

The POS should be added in its alphabetical place and the number at the start of the list must be increased (by 1) to reflect the new number of POS. The exact spelling of the Gentrack name for the POS must be entered exactly (case insensitive – everything is converted to capitals internally in the software).

5. In the file define.h, the POS must be added to the (vertical) list of POS headed by the comment "// POS :" at the same point of the array as in the above step.

```
// POS :
#define All          1
#define Albury       2
#define Balmoral     3
etc...
```

FIGURE 22 CODE FRAGMENT – POS DEFINITIONS

The numbering should then be changed to reflect the addition. This will involve increasing the number beside each POS in the list after the new one.

6. If the new POS has more than one feeder (that is being, or could be used) AND these feeders are entered into Gentrack. Add an array titled <pos name>FEEDERS to the file network.h in the same format as the existing FEEDER lists.

```
char* alburyFEEDERS[] = {"5", "All", "Albury1", "Albury2", "Albury3",
"Albury4"};
```

FIGURE 23 CODE FRAGMENT – FEEDER DATA STRUCTURE

The names of the feeders MUST MATCH the names in Gentrack, with the addition of the "All" at the front.

7. The POS and FEEDER structures are referred to in two of the software source code files. They are frontend.c, controlling the user interfaces of the program and calc.c which is the high-level engine for performing calculations.

8. In frontend.c: The following changes are only required if the new POS has feeders listed in a FEEDER structure of step (6).

9. In frontend.c, procedure SettingsDlgProc there are two changes. This procedure controls the options and actions within the Settings screen of the program. The first change occurs in the list after the comment "// use feeder index to get name of feeder". If the POS has multiple feeders, add the FEEDER array to the list in exactly the same format as the existing ones.

```
// use the feeder index to get the name of the feeder
        if (index == 1) // = "all"
           strcpy(FeederID, POS); // Same name as POS
        else
           switch(currentPOS)
           {// Depending on the POS selected, different feeder array to index
                        case Albury:
                                strcpy(FeederID, alburyFEEDERS[index]);
                        break;
```

FIGURE 24 CODE FRAGMENT – SELECTING APPROPRIATE FEEDER ARRAY

10. In frontend.c, procedure SettingsDlgProc the second change occurs after the comment "// now, depending on what POS is, reload feederID box". This will load the names of the new feeders into the feeder list if the new POS is selected. Add the FEEDER array to the list in exactly the same format as the existing ones

```
// Now, depending on what POS it is, reload feederID box
switch(currentPOS)
{
    case Albury:
        items = strtod(alburyFEEDERS[0], &ptr);
        for(index=1; index < items+1; ++index)
                SendDlgItemMessage(hDlg, DB_FeederID, CB_ADDSTRING, 0,
                                (LPARAM) (LPCTSTR) alburyFEEDERS[index]);
    break;
```

FIGURE 25 CODE FRAGMENT – LOADING FEEDER NAMES INTO LIST

11. The last change occurs in the file calc.c, in procedure setupFeederCalc. This procedure controls what happens if the "All" feeders option is selected for a POS in the Settings Screen. If the POS has feeders, then these must be calculated separately; otherwise the name of the POS is the name of the only feeder off that POS.

    Under the command switch(posNUMBER) and the comment that starts "// Special case –": If the POS DOESN'T have listed feeders under step (6), put the name of the POS in the top list. Else, if the POS HAS listed feeders, put the name in the bottom list.

```
switch (posNUMBER)
{
    // Special case - if there are no (extra) feeders off this POS
    // eg. Balmoral, Haldon
    //  Just pass POS name through to calculating procedure
    case Balmoral:
    case Fairlie:
    case Geraldine:
    case Glentanner:
    case Haldon:
    case Pareora:
    case PlPt:
    case Unwin:
            calcAVGEfeeder(FeederID, U15, P15, hMsg, MasterLIST);
    break;

    case Albury:
    case GrasmereSt:
    case HuntSt:
    case Studholme:
    case Tekapo1052:
    case Temuka:
    case Timaru:
    case Twizel6312:
    case VictoriaSt:
            // Get name of segment that this POS feeds
            POSsegment = Site2Seg (FeederID, MasterLIST);
            etc …
```

FIGURE 26 CODE FRAGMENT – CALCULATION METHOD SWITCHING

12. With the above tasks completed, the program should be re-compiled and will now work with the new POS structure.

Appendix E: Glossary of Terms

ALP	Alpine Line-charge Program – The software package created by this project
C	C and C++ are programming languages used to create software packages. In this project Borland C++ v4.5 was used to create the software. However only the standard C language aspects of the program were used at the request of Alpine Energy.
ENS	Electrical Network System – The Gentrack database that stores the physical structure of the network.
Gentrack	This is the program (and database) that models the physical structure of the Alpine Energy distribution network.
GIGO	Garbage in, garbage out
GIS	Graphical Information Systems. A method of using a computer to overlay (multiple) databases on a computerised scale map. This allows the data to be indexed geographically.
HV	High Voltage. Normally 11,000 or 33,000 volts (11kV or 33kV)
kVA	kilo-VA. VA is a measure of capacity. It stands for Volts*Amps – and is similar to watts. A normal house is said to have a 15kVA capacity.
kV	kilo-volt. 1,000 volts.
kWhr	kilo-watt hour. A measure of energy consumed. Equivalent to using a 1kW bar heater for an hour, or a 2kW heater for 1/2 and hour, etc.
line	An electrical transmission line.
LV	Low voltage. Normally 415V (230V)

ODV	Optimal Depreciated Value. A method of valuing network assets (mainly lines). This is the cost to replace the current asset with the best choice available today (this may not necessarily be the same as the existing asset). The cost of the new asset is depreciated by the age of the existing one.
POS	Point of Supply. A point in the network that connects to Transpower (the national grid) and supplies electricity to an area.
RC	The replacement cost of an asset (normally a line)
REP	The replacement cost of an asset (normally a line).
ROI	Return on Investment. An economic term for calculating the revenue required from and operation based on a percentage of the capital value of assets employed.
SOLEC	Separation of line and energy charges. As a result of the deregulation of the industry, companies had to define exactly where the revenue collected was being allocated.
TOU	Time of Use. This is an alternative method for large users of electricity to pay for it. The rate of consumption is metered in 1/2 hour intervals and the price per unit changes over the period of a day.
TX	Shorthand for transformer.
TXD	Gentrack ENS shorthand for a Distribution Transformer. Converts 11kV (HV) to 415V (LV) (230V) for household use.
Y2K	Acronym for the year 2000 bug in which computer software (and hardware) fails to accept the change from 1999 to the year 2000.

Appendix F: INDEX OF TERMS

F

G

I

L

O

P

R

S

U

Y